RasPi

DESIGN
BUILD
CODE

13

Get hands-on with your Raspberry Pi

WHY YOU NEED

PYTHON 3

Set up **AirPlay** Speakers

Plus Make a digital photo frame

Welcome

The big focus this month is Python 3. You see, despite the fact that vast numbers of us code in the deprecated Python 2.7, we do need to switch to the more modern yet often neglected version of the language. One of the biggest reasons is the upcoming end-of-life date for Python 2.7, and the fact that the devs are very clear about there not being a version 2.8. So it's time to get up to speed, and this month we give you a flying start. Sticking with the Python theme, we're also looking at how to optimise your code to make it run faster on the Pi. Plus, we have a cool selection of projects for you to try out this weekend, too – including a digital photo frame and a wireless stereo system. Enjoy!

Gavin Thomas

Editor

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

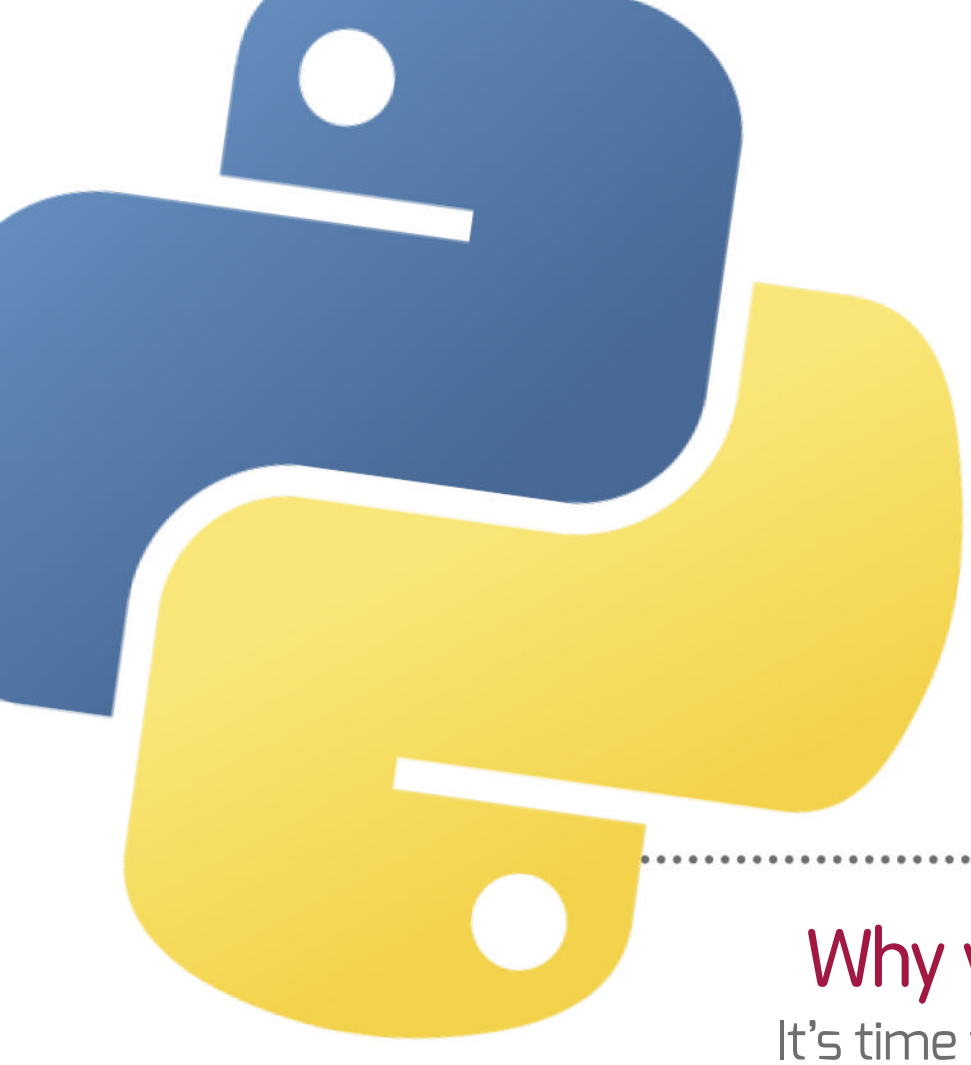
Learn to make and code gadgets with Raspberry Pi

From the makers of

LinuxUser & Developer

Join the conversation at...

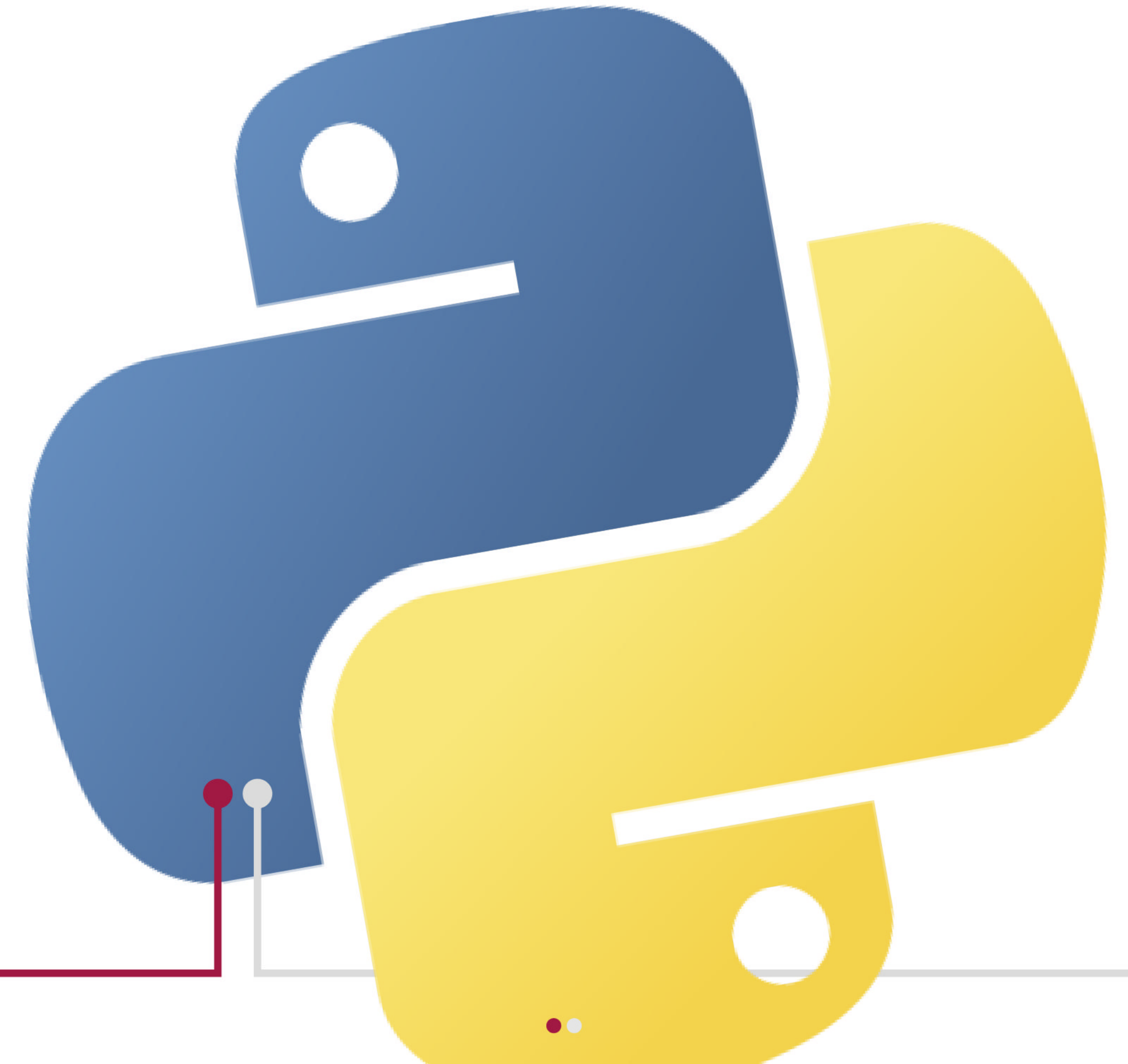
@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk



Contents

Why you need Python 3

Bring your programming skills up to date by finally letting go of Python 2 and embracing the better language



Python 3 has been languishing. There have been many good reasons for this, including a lack of dependencies, monolithic code bases written in Python 2 that are costly to port across, and also the prevalence of tutorials such as those found in this magazine that continue to focus on the outdated 2.x versions, among many other reasons. One of the common threads uniting all these reasons not to use Python 3 is a sense that there aren't enough people actually programming in Python 3 to help spread its adoption. A few years ago (we've had Python 3 for seven now), things like the lack of dependencies were real inhibiting factors, but the situation has vastly improved since then and some of these reasons just don't hold up any more.

So why haven't people switched yet? Part of the problem is perhaps down to the dual support of both Python 2 and Python 3 by its developers, but in any case there is no real reason nowadays to continue writing code in the deprecated version of Python. Momentum is needed from the community in order to successfully make the leap from Python 2 to the much, much better Python 3, and to ensure that Python isn't abandoned altogether in favour of Ruby or Go, for example.

The adoption of Python 3 by the community is something that we personally believe in here at **RasPi** and, since we regularly write Python tutorials for you, it makes sense for us practise what we preach. We won't throw you into the deep end immediately, but over the coming months we are going to begin transitioning across to using Python 3 ourselves. When you download your tutorial assets each issue, keep an eye out for the Python 3 files that we'll be adding. For now, though, read through our Python 3 primer and we'll get you started with this fantastic programming language today.

"Momentum is needed from the community in order to successfully make the leap from Python 2 to the much, much better Python 3"

Getting started

Let's take a look at the new print function and handling lists

There are a load of little changes that have made their way into Python 3. Depending on the version of Python 2 you're using, some of these changes will be more drastic than others – Python 2.5 and onwards received some function changes backported to the code, however there are still some notable differences. This happened quite a while ago though, so we'll be focusing on going from current Python 2 to Python 3.

> "One of the changes made in Python 3 that will cause a stumbling block to some is the update to the print command"

Say hello

One of the changes made in Python 3 that will cause a stumbling block to some is the update to the **print** command. Before, it was fairly unique as a statement and not a proper function, allowing you to string together outputs for debugging or just normal outputs.

You can still do this, but to bring it more in line with the rest of the way Python works, it's now a proper function: **print ()**. You can perform the exact same kind of functions that you could with the original print, albeit formatted somewhat differently. To illustrate this, let's first of all create a Hello World command. Open up IDLE 3 and type this into the shell:

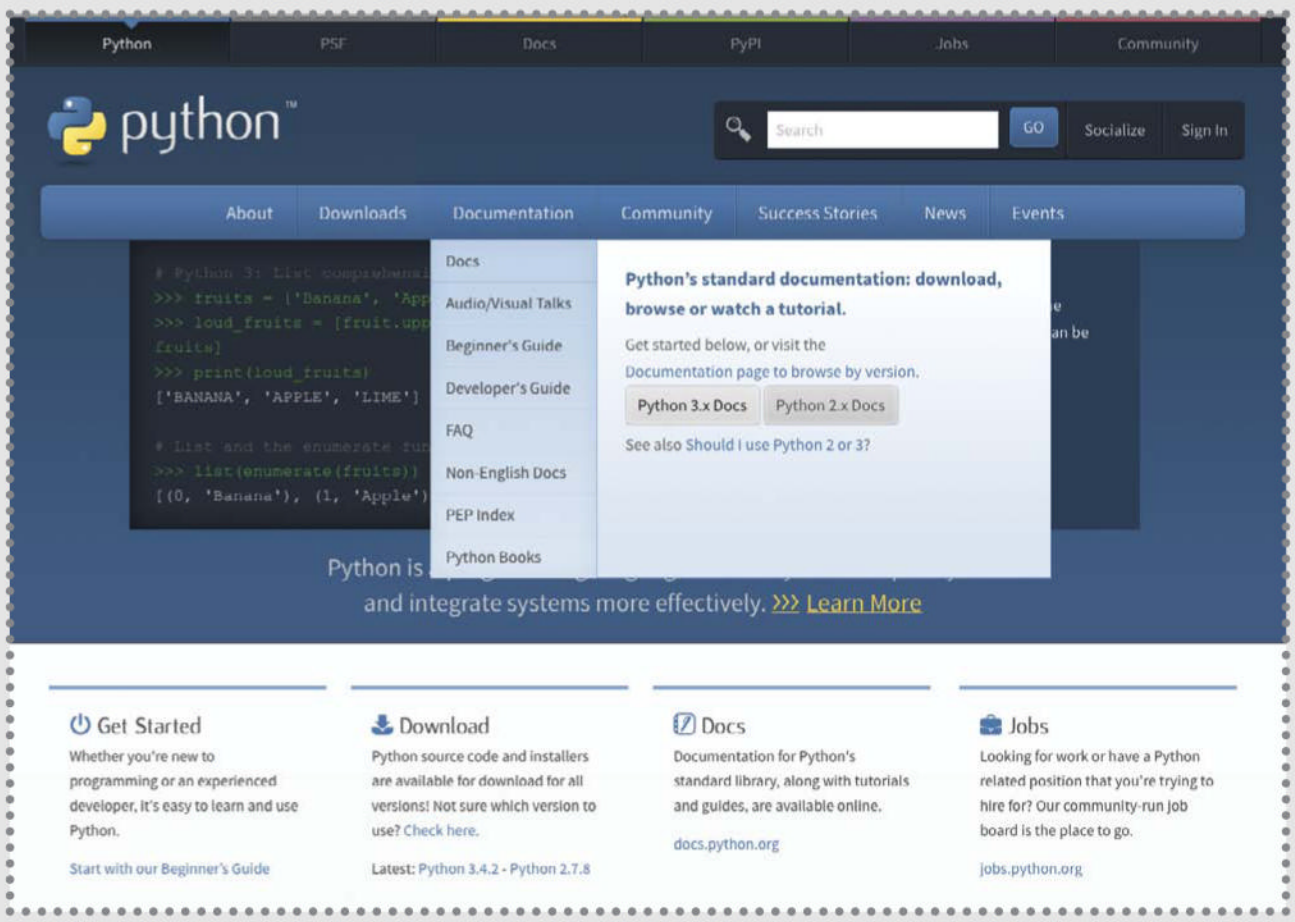
```
print (“Hello World”)
```

Very simple. For more complex strings that include variables and maths, it’s also quite easy to create them as well. For the next example we’ll create a variable with a number to illustrate how it works. Back in IDLE, use the following two commands in the shell:

```
a = 5
```

```
print (“You should know that ”, a, “ is
greater than ”, 2*2 , “ every time”, sep=“”)
```

As expected, this creates the sentence: “You should know that 5 is greater than 4 every time”. We’ve thrown the **sep** argument on the end of the print function; this allows you to set what’s used to separate the text and inserted variables. In this case we’ve told it not to add anything, but we could tell it to add a space and cut down on the length of the line.

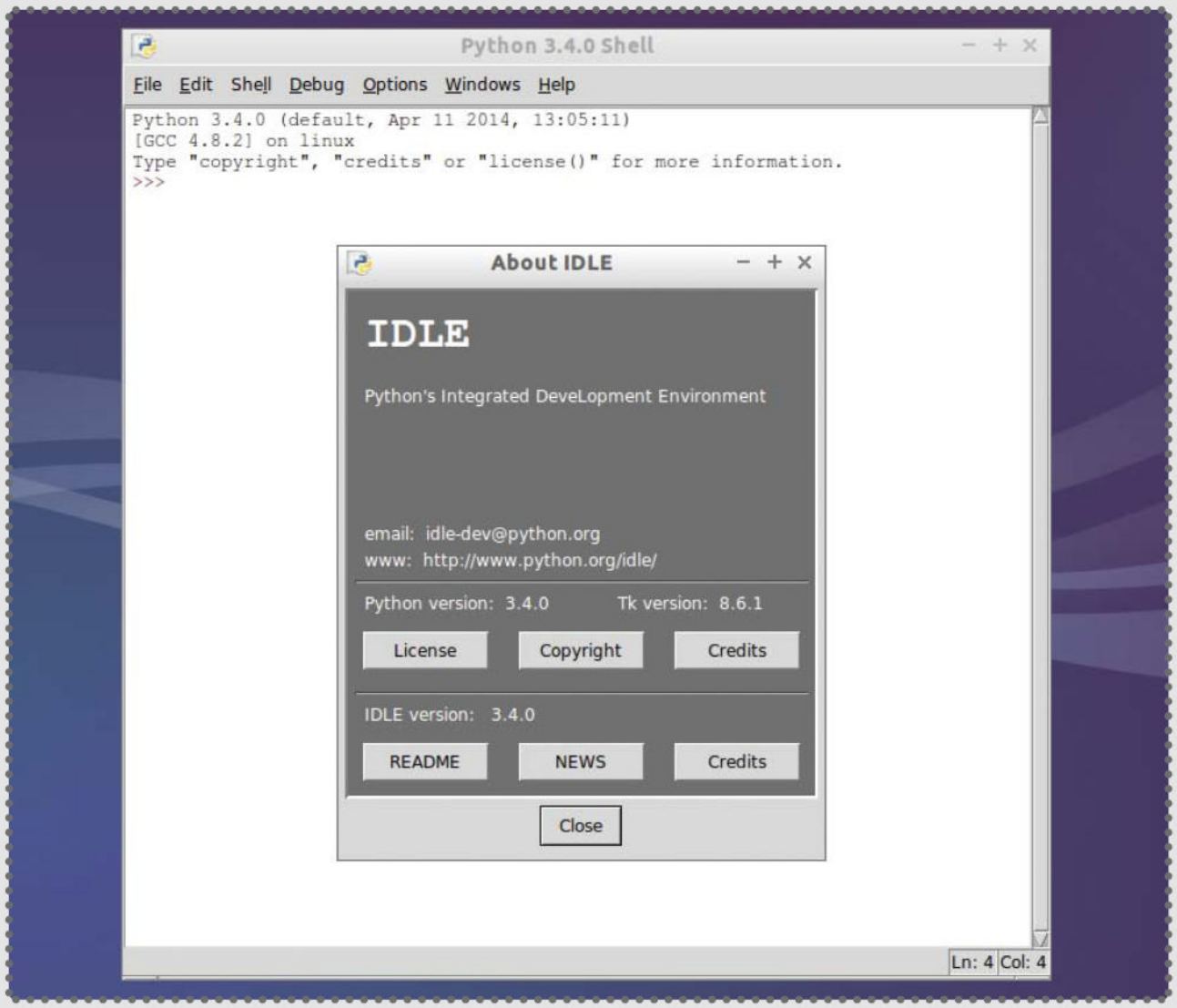


Why you need to change

Python 2 is slowly dying. For the past few years it’s had very little active development beyond bug fixes and the odd bit of deprecation. We constantly run into code bits that can be dealt with very easily in Python 3, mainly involved with sorting lists and databases, and in general it’s more in line with the Python philosophy of being easy to read and use, as shown with the changes to things like print and input.

There’s a bit more to the differences than what we’ve managed to show in this article, however these are some of the biggest and most common functions that are used in Python, so getting the very basics down is important. We’ll discuss where you can get more information about the changes later on and supply links to documentation as well.

Left It’s worth reading the ‘Should I use Python 2 or 3’ article on **python.org**



Left Check which version of IDLE you're running by opening the About dialog

On the list

Standard lists are handled basically the same way in Python 3 as they are in Python 2. The usual **list.append()** and **list.remove()** commands are still in place, but Python 3 now includes a couple of new commands to make organising them much easier:

list.clear() – this simply removes all the items currently in the list and it replaces **del a[:]**.

list.copy() – this prints out a basic copy of the current list, similar to if you used **print (list)**. This replaces **a[:]**.

On the other hand, mappable objects such as dicts work a bit differently. Common commands such as **dict.keys()** and **dict.items()** return views instead of lists. These are dynamic and update as the list's entries are changed.

Python 3 setup

A few distros will have Python 3 pre-installed, but most will have Python 2.
It's not difficult to get the new libraries set up. In most instances, you will need to look for the python3 package in your system's software repos, however in Arch and Fedora it's currently listed as python. You'll also need to install the correct version of IDLE, Python's IDE. This will be in the repos under idle3, or failing that as plain idle if python3 is python in your repos.

On some distros, IDLE for both Python 2 and 3 may be installed side-by-side. Make sure you're using the right IDLE – it won't damage anything, but it can be frustrating trying to bug fix code when it's not even running properly.

The Code PRINT FUNCTION

```
#!/usr/bin/env python3

# The above is a slightly different shebang to let the
# interpreter know which version of python to use when
# running it from terminal or command line. In previous
# tutorials we have had it end in python2 for just these
# sort of occasions.

# Let's create some variables and lists to play around with:

hello_world = "Hello World!"

hello_list = ["Hello", "there", "world", "!"]

hello_dict = { "first" : "Hello",
               "second" : "World",
               "third" : "!" }

# Here's how we can print them all out, first the normal print:

print (hello_world)

# Then print out the entire list in order. The asterisk
# allows for each item to be called individually rather
# than the list

print (*hello_list)

# Finally, print the dictionary

for x in hello_dict:
    print (x, ":", hello_dict[x])
```

The full function

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

sep – sep enables you to change what is used to separate parts of the output. By default it's a space, but you can change it to any compatible character. This can be used to make the line shorter, or include mathematical operator such as >.

end – how the end of the line is portrayed. To leave a space rather than start a new line (which is the default), use something like print (x, end = " "). Like sep, this needs to be a string and will not be converted into one like other outputs.

file=sys.stdout – this completely replaces the kind of command where you would write print >>sys.stderr, "an error". Construct the string as you would normally for anything you're printing, then append the file option like so: print("fatal error", file=sys.stderr).

flush – forcibly flushes the print stream after execution if True.

Reading input

Now you've got the basics down, get up to speed with using human input and variables

Need input

Human input via text or data may be needed for your code, and this has been changed slightly in the current version of Python. Whereas before you may have used **raw_input** to get your data, it's now been switched to simply **input**. It works just about the same as before, and is still quite simple. Here's the description for it:

```
input([prompt])
```

So in order to get it to work, you just need to do the following inside the shell:

```
a = input()
```

You'll be able to enter a string of text or a number as usual, and calling the a variable will print out the result. You can still do similar things with the input command, such as print a message for the user to read to indicate what kind of input is required. This can be done with something like:

```
b = input("What's your name? ")
```

"You can still do similar things with the input command, such as print a message for the user to read to indicate what kind of input is required"

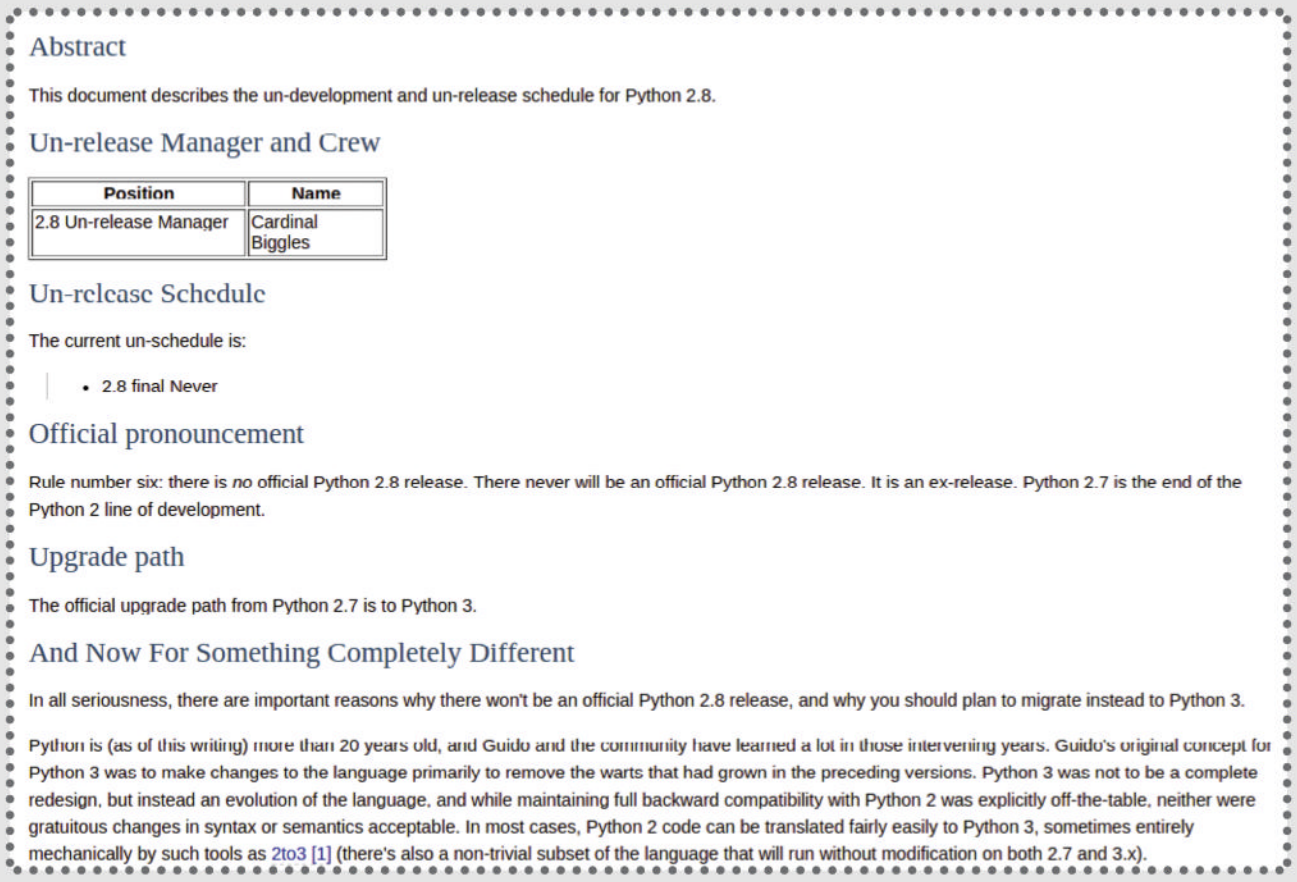
The space at the end separates the prompt from the input field like it did in Python 2, and is just a nice way to format the command. You don't have as much control over what the text in the input can be compared to with the print command, so you're better off doing a mix of print and input, making use of the print command's **end=" "** option so it doesn't create a line break if you don't want it.

Take a rest

A new rest function in Python 3 exists that allows you to create iterable lists from part of a range. It's easiest to explain in an example, which works as follows:

```
(a, *b, c) = range(5)
```

The variable a is given the value 0, c gets 4 and b becomes a list of [1, 2, 3]. The asterisk is the key part here and lets Python know to put the rest of the iterations into the variable as a list. This doesn't need to be done between variables



Abstract

This document describes the un-development and un-release schedule for Python 2.8.

Un-release Manager and Crew

Position	Name
2.8 Un-release Manager	Cardinal Biggles

Un-release Schedule

The current un-schedule is:

- 2.8 final Never

Official pronouncement

Rule number six: there is *no* official Python 2.8 release. There never will be an official Python 2.8 release. It is an ex-release. Python 2.7 is the end of the Python 2 line of development.

Upgrade path

The official upgrade path from Python 2.7 is to Python 3.

And Now For Something Completely Different

In all seriousness, there are important reasons why there won't be an official Python 2.8 release, and why you should plan to migrate instead to Python 3.

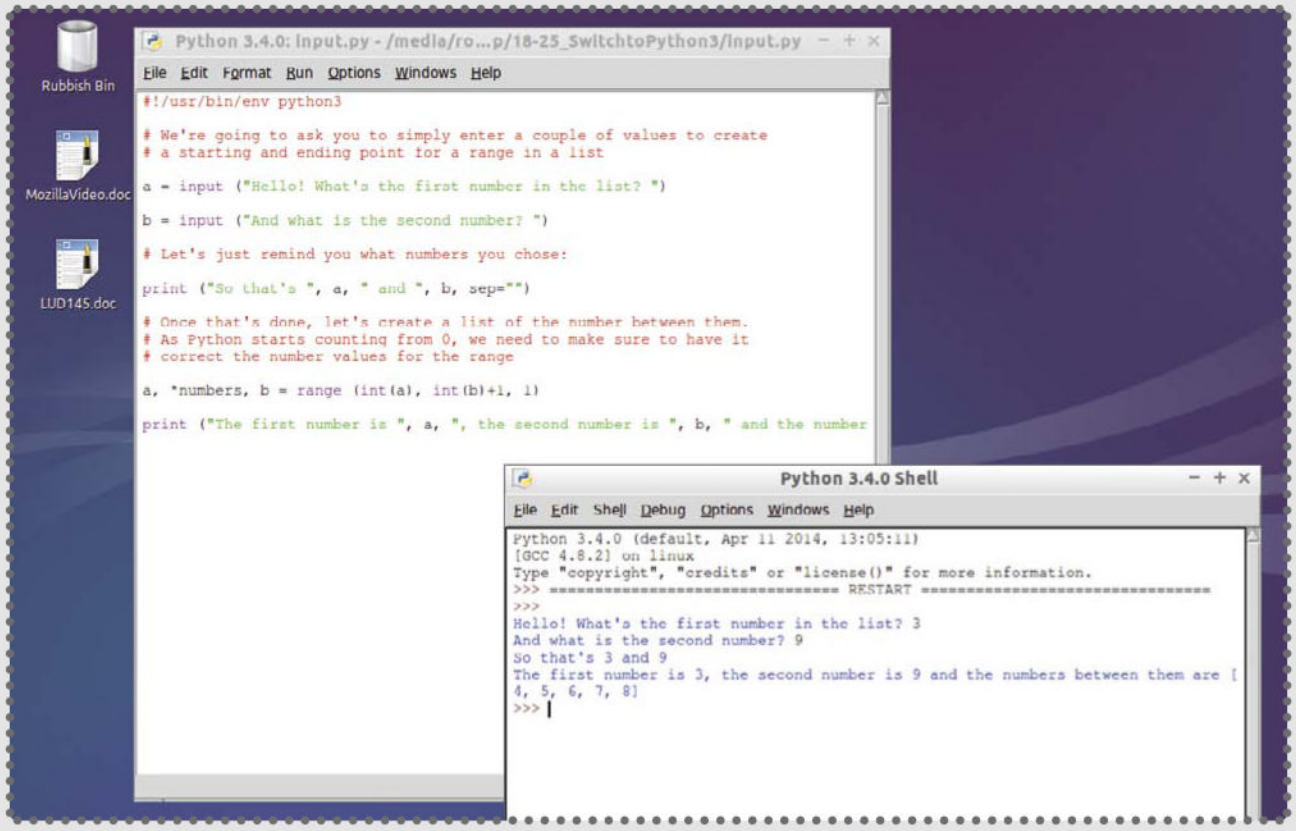
Python is (as of this writing) more than 20 years old, and Guido and the community have learned a lot in those intervening years. Guido's original concept for Python 3 was to make changes to the language primarily to remove the warts that had grown in the preceding versions. Python 3 was not to be a complete redesign, but instead an evolution of the language, and while maintaining full backward compatibility with Python 2 was explicitly off-the-table, neither were gratuitous changes in syntax or semantics acceptable. In most cases, Python 2 code can be translated fairly easily to Python 3, sometimes entirely mechanically by such tools as 2to3 [1] (there's also a non-trivial subset of the language that will run without modification on both 2.7 and 3.x).

End of life

In the past year, a time frame to make the switch to Python 3 has become a bit more urgent, as the end of life for Python 2.7 has been announced. Python 2.7 itself is the last version of the Python 2.x line, with firm plans to not release a Python 2.8 (going so far as to have a wiki page with an 'un-release schedule' for it).

Python 2.7 will stop receiving bug fixes and other updates in 2020, so you still have five years to get your code ported over to Python 3.0. As for Python 4.0, that's not going to happen any time soon so you should be fine to safely port to 3.0 without worrying about a massive jump in the future.

Left Python 2.7 is still hugely popular, which is why its EOL delayed from 2015 to 2020



Left This simple implementation of the rest function shows how useful it can be

either, and can be used at the end or front of a function to create a list from the rest of the iterations, as shown here:

```
a, *othernumbers = range (3)
*firstnumbers, last = range(11)
```

This will result in a = 0, othernumbers [1,2], first numbers [0,1,2,3,4,6,7,8,9] and last as 10. These types of separations can be used for while loops to determine when to finish a particular task, or other similar functions. In terms of the while loop you can do something like this:

```
*listofthings, iterations = range (6)
loops = 0

while loops < iterations:
        listofthings[loops] = “hello”
        loops += 1
```

This creates a list of five hello’s, but you can easily use it for more practical functions depending on your specific needs.

“These types of separations can be used for while loops to determine when to finish a particular task”

The Code READING INPUT

```
#!/usr/bin/env python3

# We're going to ask you to simply enter a couple of values to
# create a starting and ending point for a range in a list

a = input ("Hello! What's the first number in the list? ")

b = input ("And what is the second number? ")

# Let's just remind you what numbers you chose:

print ("So that's ", a, " and ", b, sep="")

# Once that's done, let's create a list of the number
# between them. As Python starts counting from 0, we need
# to make sure to have it correct the number values for
# the range

a, *numbers, b = range (int(a), int(b)+1, 1)

print ("The first number is ", a, ", the second number is ",
       b, "and the numbers between them are ", numbers, sep="")
```

Easy conversion with 2to3

So far we've shown you how to write code in Python 3, focusing on the changes that have been made to the structure of Python. You can apply this to code you've already written using Python 2 to manually change it to work with Python 3, however the larger the files the longer this will take. There is a slightly easier and more automatic method that you can use called 2to3.

Simply, 2to3 is a script that converts Python files written in Python 2-style code into Python 3-compatible code. It should be available as long as the Python 3 libraries have been installed to your system. To get it to work you just need to run something like:

```
$ 2to3 -w file.py
```

This changes the file completely; otherwise you can leave out the -w option so it creates a copy instead. This is not a fool-proof conversion though, and you'll likely need to do some bug-fixing.

Putting it together

Here's a short piece of Python 3 code that illustrates everything we've discussed so far

Let's go through this line by line. We import time so we can run the program at a steadier pace. Importing is done in the same way as Python 2, and most of the standard modules are still in place:

```
import time
```

This creates our basic variables – a string for repeating the code and a list to store the sentence we're making in this bit:

```
repeat = "y"
sentence = []
```

Defining functions will work the same as before. You can still pass variables to it from your work, but here we're opting for the global function:

```
def textinput():
    global sentence, repeat
```

We want to create a sentence of a specific length and we then turn that into an integer so we can use it elsewhere.

> "Importing is done in the same way as Python 2, and most of the standard modules are still in place"

> “The Python 3 documentation contains basic explanations for all the built-in functions and standard modules”

The iterations variable is used to make sure we create a sentence of the appropriate length:

```
    sen_length = input (“How long do you
                  want this sentence to be? ”)
    sen_length = int(sen_length)
    iterations = 0
```

This while loop will let us append each word individually to our custom sentence. It counts the iterations to know when to stop:

```
    while iterations < sen_length:
        print (“What do you want want ”,
          iterations + 1, “ to be?”, sep=“”)
        sentence.append(input())
        iterations += 1
```

As in a previous example, we’ll use an asterisk to print the sentence:

```
    print (*sentence)

    time.sleep(1)
```

The repeat variable is changed so that we know whether to loop or stop the code:

```
    repeat = input("Do you want to try again?
                        (y/n): ")
```

Some basic printing which will introduce the code:

```
print ("Hello, and welcome to our Python 3
code example!")
```

```
time.sleep(1)
```

```
print ("Let's create some text.")
```

This while loop is the main part of the program, running the function above and also determining whether to continue the code or not:

```
while True:
    time.sleep(1)
    if repeat == "y":
        textinput()
    else:
        print ("Goodbye!")
        break
```

Resources

Want more? Check out these for help and documentation on Python 3

What's new in Python 3.0

https://docs.python.org/3/whatsnew/3.0.html

One of the best places to learn a majority of the little changes made to Python 3.0 over both 2.0 and 2.5+ is the documentation created by the Python folks themselves. This covers the basics such as common missteps, most of which are covered in this article, along with a lot of the tiny things you'll only come across now and then.

If you start getting the odd syntax error or unknown result, you may need to start looking through the information here to figure out if your use of != and == is to blame, or whether you're using now-reserved words like True and False.

The Python 3 documentation

https://docs.python.org/3/contents.html

If you didn't know Python inside-out even while using version 2, you can always refer to the documentation for the language. It contains basic explanations for all the built-in functions and standard modules, usually along with some examples to illustrate briefly how they work. They will also usually let you know if any other functions are related to them and you can click through to them.

"If you didn't know Python inside-out even while using version 2, you can always refer to the documentation"

This is very useful as your first port of call for when you're trying to figure out why a particular piece of code isn't working, and can even give suggestions on how to improve it or pair it with something else.

Stack Overflow

https://stackoverflow.com/

If you have a particularly tricky bit of code that just won't work and there's no solution to be found on the documents, Stack Overflow is the best place to go. The question and answer site is free and tags easily sort the questions so that you can find questions on Python easily. Either someone else will have run into the same problem, or you can ask about it and get a solution usually very quickly. It's platform agnostic, which is fine because so is Python, but it also means you get more people looking over queries.
The website operates on a reward system for correct answers, so people are always willing to pitch in and help out others.

LinuxQuestions

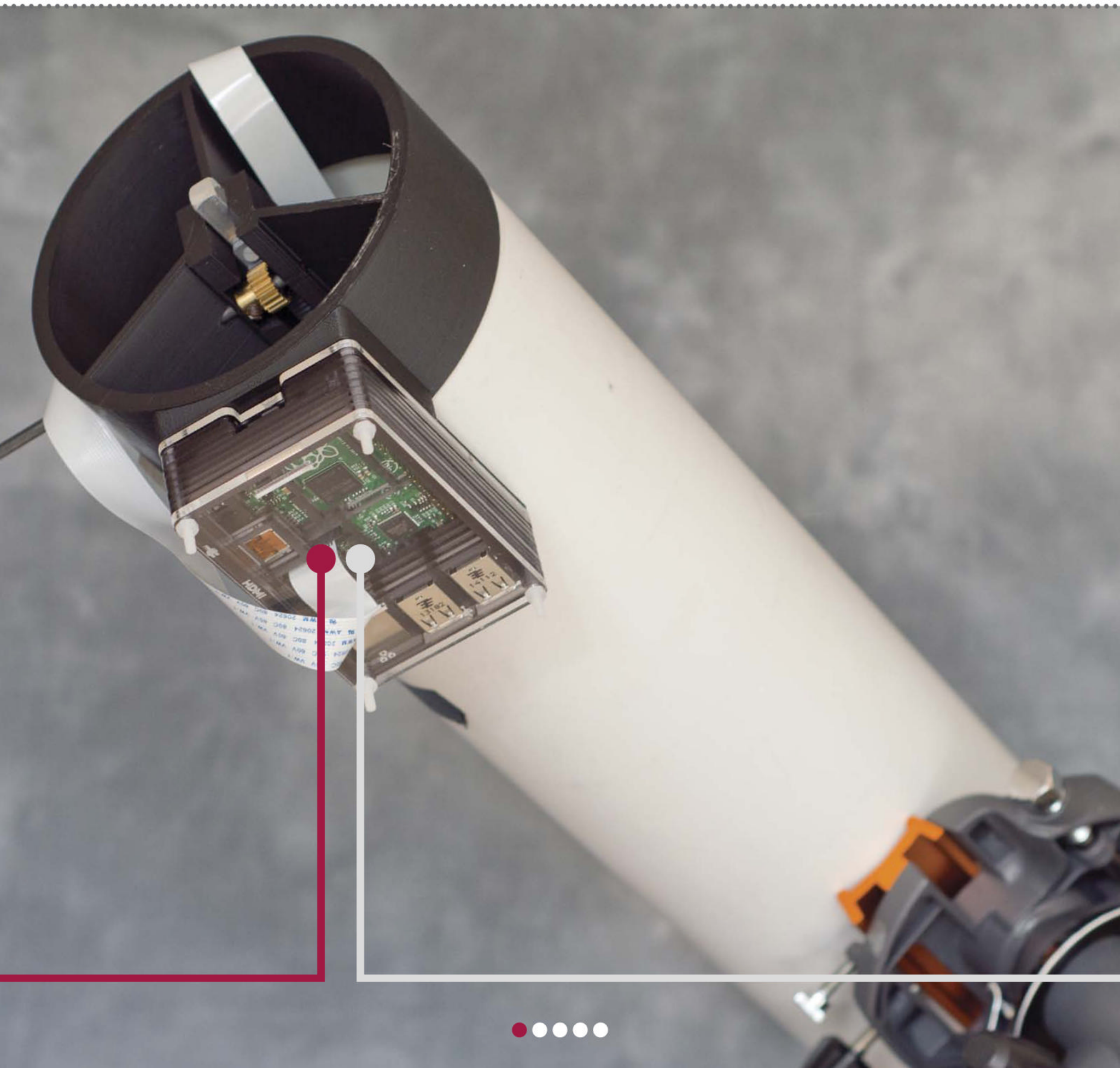
https://www.linuxquestions.org

Some problems might come down to Linux itself, so as a last resort you can always check the LinuxQuestions forum for advice on how to either fix your code or fix your system to run it properly. Make sure you search for your question first, in case someone else has come across this problem (it's also part of forum etiquette to look it up first before asking). The folks here will also be able to help you with your code, but the base of users is slightly smaller than that of StackOverflow.

"If you have a particularly tricky bit of code that just won't work and there's no solution to be found on the documents, Stack Overflow is the best place to go"

PiKon

Mark Wrigley and Andy Kirby take photos of the Moon using a 3D-printed telescope and the Pi camera module



Tell us how you began your project

Mark Wrigley I run this small company called Alternative Photonics and like to find out about new technologies. When Sheffield's Festival of the Mind came along I thought of what could we put together in terms of things that are new and disruptive, and that's how the whole project started. The two things I was interested in were using Raspberry Pis for photography and 3D printing. With 3D printers you've got a device in the ballpark of £500, putting it amongst the price you'd pay for consumer items, so manufacturing is actually something you can do at home. There are companies who will print something for you, so you send in a design and they'll produce it. And there are maker communities – Andy runs a group called Sheffield Hardware Hackers and Makers.

Andy Kirby Sheffield Hardware Hackers and Makers is for anyone off the street to come along to if they want to hack and make things. I set it up as part of the original RepRap project that ran on the Internet quite some years ago [Ed: RepRaps are 3D printer kits that can print parts for more RepRaps]. I found that there were quite a few people building printers who got to a certain point and got stuck, so I set up this group originally to be a drop-in for people to come along to and get the help they needed.

Andy, what was your role in the PiKon?

Andy I helped Mark pick a 3D printer that was going to be pitched right for his skillset and then, as he was building up the printer, when he got to bits where he got stuck he'd bring it along to the Hardware Hackers group and say 'It doesn't do this right' or 'I can't get it to do that', and we'd work through the problems together. Once he'd got his printer going, I worked through the CAD software with him to see what was available that was open source and

Mark Wrigley is a member of the Institute of Physics and holds a Licentiateship with the Royal Photographic Society

Andy Kirby was an early contributor to the RepRap project and runs the Sheffield Hardware Hackers and Makers group

within his capabilities. There are various things you can't do with a 3D printer when you design parts, so we had a conversation about that and I gave him the shortcut to get working parts out quicker.

How many different parts need to be printed and assembled?

Mark There are five printed parts, and then there's one piece of white venting duct (which forms the barrel), a couple pieces of Meccano, a small square piece of aluminium and the focusing knob. We had a think about printing the telescope tube but it would take an awful long time to print sections that were long enough, so we went for something that was reasonably available. We had lots of discussions about cardboard tubes and so on, and we came across a bit of white venting duct that worked.

How does the PiKon work?

Mark Most telescopes work on the principle that you have some sort of object lens or mirror which forms an image and then you use an eyepiece to examine that image, so it's usually what's termed as a virtual image. With the PiKon, what we're doing is using the objective device, in this case a 4.5-inch (335mm) mirror, to form an image and then placing the Raspberry Pi camera without its lens – so just a sensor – where the image is formed. So effectively, instead of using an eyepiece we're examining the image electronically by placing the Raspberry Pi sensor there, and the sensor is suitably small so we're able to examine a fairly small area of the image.

What kind of resolution do you get?

Mark At the moment, the setup that we've got is equivalent to a magnification of about 160. If you look at

If you like

The Raspberry Pi Foundation website featured a project that mounts the Pi and camera onto a telescope and captures great images:
https://bit.ly/1qTp3Pb

Further reading

If you're interested in astrophotography and developing software for PiKon, check out these astronomy packages:
https://bit.ly/100wj65

the Moon, the field of view of your eye is about half of one degree; the PiKon's maximum field of view is about a quarter of one degree, so effectively you can see about half the Moon in full frame. The next thing I'd like to do is look at planets. In terms of its resolution, the PiKon's sensor is five megapixels, which is 2500x2000 pixels. If you're going to reproduce an image in print you'd normally use something like 300dpi, so 5MP would allow you to reproduce something like an A4 image. On a computer screen all you need is 72dpi, so what I'm quite interested in doing next is seeing how much magnification we can get simply by cropping – so literally throwing away some of the pixels to be able to zoom in on something like a planet. If you read the Astronomy For Beginners stuff, they talk about needing a magnification of 200-250 to be able to observe planets, so I'm hoping we can do things like see the rings of Saturn. We're not out to rival the Hubble – we think that what you've got is a reasonable instrument and you can do a few interesting things with it. The other thing is that because it's using a Raspberry Pi sensor instead of an eyepiece, you're immediately into the world of astrophotography rather than doing observations, so that's the sort of way we're going.

How do you control the PiKon's camera?

Mark We would like to do it better!

Andy At the moment it's done through the command line. I'm not a Raspberry Pi programmer… So we're using raspistill at the moment, which gives you a certain number of seconds of preview and then takes the picture, so it's a bit clunky. I'm talking to a guy who's into Raspberry Pi and is also a photographer, and he's written some programs where you have a shutter button. The next thing to do

then is to control PiKon from an input/output device like a shutter button and then give the JPG files you produce a sequential or a date-stamped filename. One thing I'd like to see next is if we could get this hardware out into the public and attract people to actually come along and develop more and more software for it. I tried taking pictures of the International Space Station with an ordinary camera, for example. It's really difficult because suddenly this dot comes flying across the horizon and you have to swing around, get your camera lined up, press the shutter and so on. One thought I had was it would be nice if you could take multiple shots with the PiKon – you press a button and then it keeps taking a new photograph every five seconds.

Below Here is an example of the kind of photo that can be taken with the PiKon telescope: the Moon, with enough detail to be able to see its craters and the surface texture

Make a digital photo frame

Take your Raspberry Pi, HDMIPi and Screenly and turn them into a beautiful digital photo frame

Digital signage is a huge market in modern times, and increasingly we are seeing digital advertising in the public space – be it on street billboards, shopping centres and even in some city taxis. Screenly is an exciting piece of software from Wireload Inc that has set out to reduce the barriers to entry of the digital signage market by employing a Raspberry Pi as its main hardware component for the individual signage nodes. With the powerful VideoCore IV graphics processor at the heart of the Pi and its low electrical power consumption, using it as a digital signage platform is a no-brainer.

Our expert has been using Screenly a lot recently for some projects and it truly is a really great piece of software. It occurred to him that, together with Screenly, it would make the perfect home-brew digital photo frame and another great Raspberry Pi-based hardware project.

THE PROJECT ESSENTIALS

HDMIPi kit

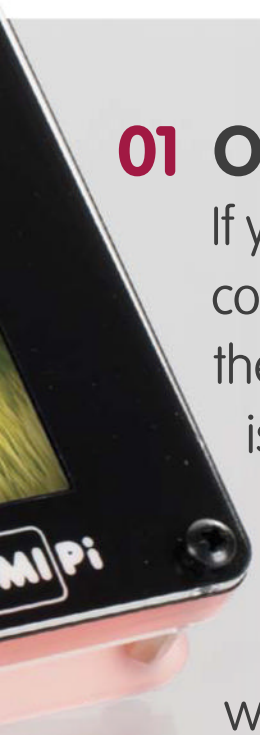
Class 10 SD Card

5.25V Micro USB power supply

USB keyboard

Wi-Fi dongle

Internet connection



01 Order your items

If you haven't already got them in your Raspberry Pi collection, you will need to order all of the items from the "Project Essentials" list at the top-right. The HDMIPi is currently only compatible with Model B of the Raspberry Pi, although an upgrade kit that makes it compatible with the Models B+, A+ and 2B was recently released. Pretty much any USB keyboard will work with Screenly, including wireless ones, so you do not need to worry about a mouse for this tutorial as it will all be done from the command line. Finally, a speaker is only necessary if you intend to play videos with sound from the display.

"It occurred to our expert that, together with Screenly, the HDMIPi would make the perfect home-brew digital photo frame"

Above The reverse view of HDMIPi, showing GPIO and connector cutouts

02 Assemble your HDMIPi

The HDMIPi comes as a do-it-yourself kit rather than a polished product. Not only does this make it cheaper for you to buy, but it also gives it a more hack-like feel in its Pibow-esque acrylic layer case. It is not hard to assemble, but in case you get stuck there is a fantastic assembly video here: **http://hdmipi.com/instructions.**

03 Download Screenly OSE

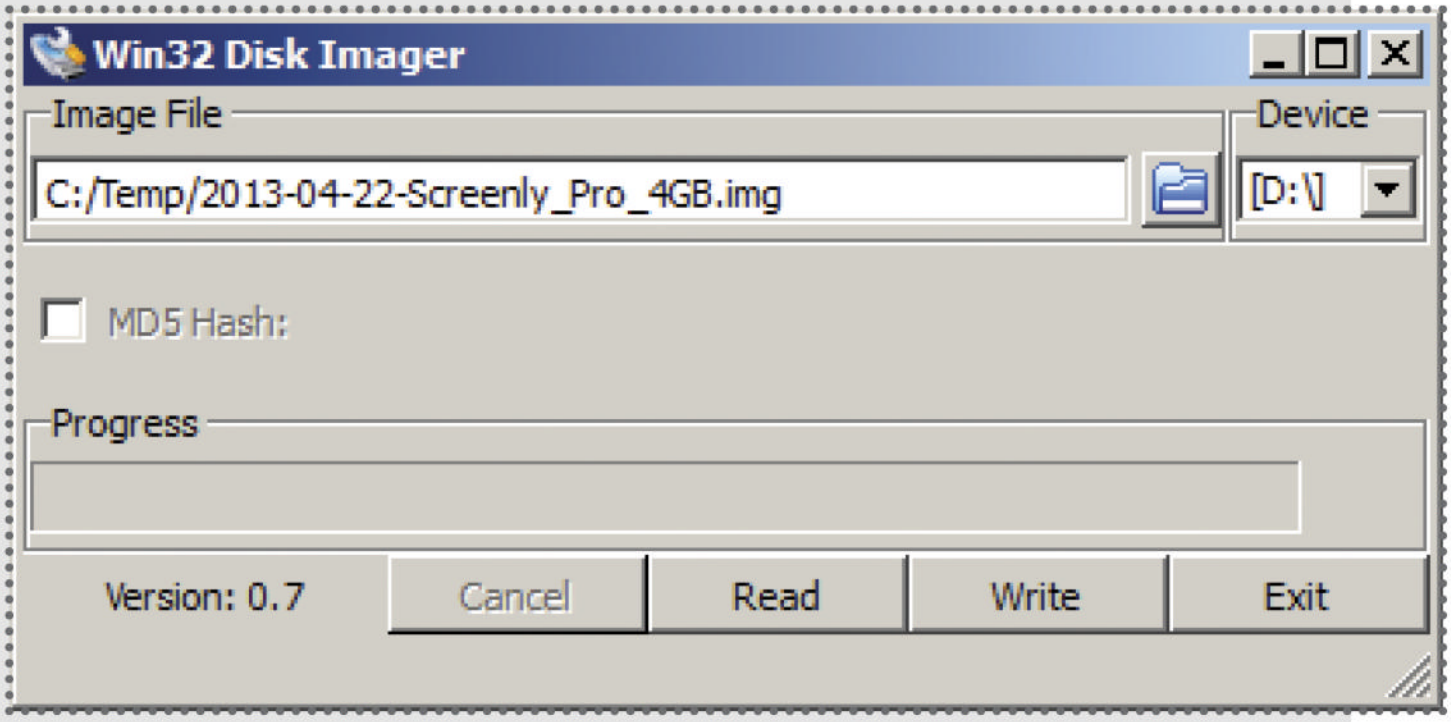
Now that you have the hardware all ready to go we need to download the Screenly OSE SD card image. This is only a 3.7GB image file, however it may not fit on some 4GB SD cards so we would recommend a minimum of 8GB, for extra space for all your pictures and videos. Visit **http://bit.ly/1wLKIRQ** and download the ZIP file from one of the mirrors under the "Getting started" heading.

04 Flash image to SD Card (Linux)

It's worth noting the value of having a Linux machine at your disposal (or a spare Raspberry Pi and SD card reader) to download the ZIP file in Step 03. This is typically the easiest way to unzip the file and copy the image across to your SD card. Assuming the disk isn't mounted, open a terminal session and type:

```
unzip -p /path/to/screenly_image.zip | sudo dd bs=1M of=/dev/sdX
```

Make sure that you replace /path/to/screenly_image.zip with the actual path.



05 Flash image to SD Card (Other OS)

If you do not have another Linux machine handy, or a card reader for your Raspberry Pi, you can still do this from other popular operating systems. On Windows you should use Win32 Disk Imager and follow the easy-to-use GUI. From Mac OS X you have the options of using the GUI-based software packages PiWriter and Pi Filler, or running some code from the command line. For more information, pay a visit to **www.screenlyapp.com/setup.html**.

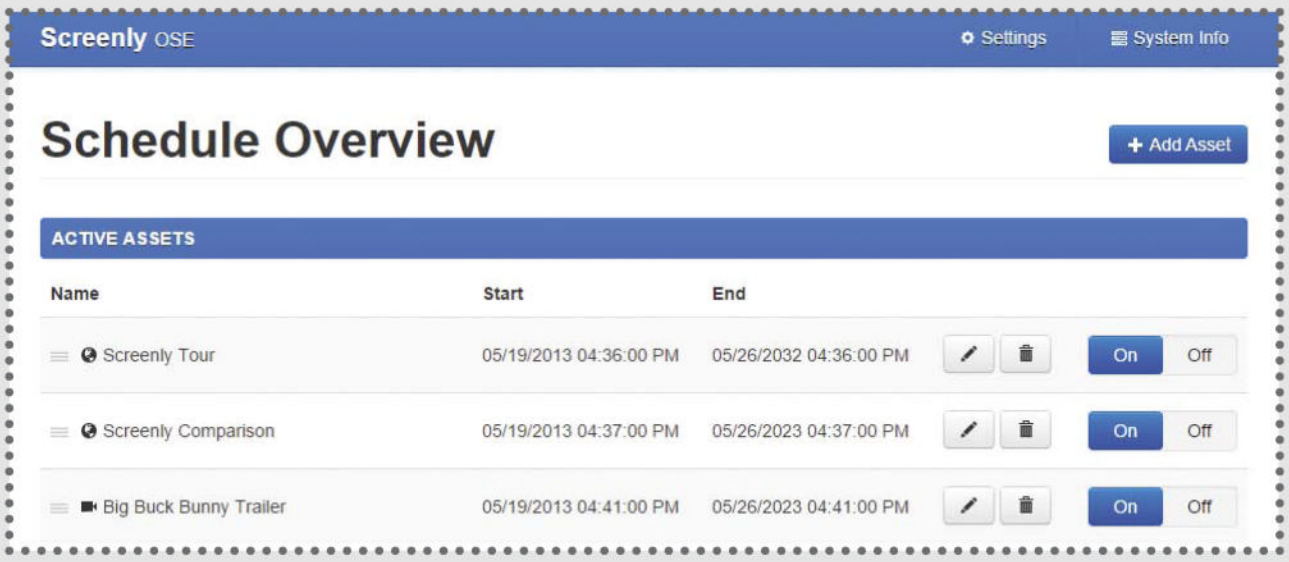
"If you do not have another Linux machine handy, or a card reader for your Pi, you can still do this from other popular operating systems"

06 Insert SD card and peripherals

Once the Screenly image has been successfully transferred to your SD card, you will need to insert it into the Raspberry Pi within your HDMIPi casing.
It is a good idea to connect your Wi-Fi dongle and keyboard at this point. Go back a couple pages and take a look at the image to see where the slots are in relation to the casing. A wired network is also required for the initial setup and for configuring the Wi-Fi.

07 Boot up your Raspberry Pi

The HDMIPi driver board has a power output for the Raspberry Pi which means you only need one power supply for this setup. Plug it in and wait for it to boot into the Screenly splash screen. An IP address (of format http://aaa.bbb.ccc.ddd:8080) should be displayed here, which will enable you to gain access to the management dashboard.



08 Disable Screenly video output

Load the IP displayed on the splash screen on the browser of a different computer – you won't be able to do it directly from the same Raspberry Pi. The Screenly OSE dashboard should now load. Once inside the dashboard, move the slider for *Big Buck Bunny* to the OFF position or delete the asset entirely.

"Load the IP displayed on the splash screen on the browser of a different computer – you won't be able to do it directly from the same Raspberry Pi"

09 Enter the command line

Once you have disabled the *Big Buck Bunny* trailer from the web interface, you should now be able to enter the command line easily and you can do this by pressing Ctrl+Alt+F1 on the attached keyboard at any time. Alternatively, you can access the command line over SSH using the same IP address as shown previously on the splash screen.

10 Run the update script

The image file we downloaded from the website is actually just a snapshot release and does not necessarily include the latest Screenly OSE code, as the code updates are made more regularly than the image. It is therefore good practice to run an upgrade to the latest version using the built-in script. You can run the following command:

```
~/screenly/misc/run_upgrade.sh
```

11 Configure Raspberry Pi

Once you are successfully at the command line, you need to type **sudo raspi-config** to enter the settings and then select '1 Expand root file system' to make sure you have access to all of the space on the SD card. Then, change the time zone (option 4 and then 12) so that it is correct for your location. If your screen has black borders around the edge you may also need to disable overscan (option 8 and then A1). We would also recommend changing the default password to something other than raspberry to stop any would-be hackers from easily accessing the Raspberry Pi via SSH (option 2). Once complete, exit without restarting by selecting Finish and then No.

"The image file we downloaded from the website is actually just a snapshot release and does not necessarily include the latest Screenly OSE code"

12 Enable and set up Wi-Fi

As Screenly runs on top of Raspbian, the Wi-Fi setup is essentially the same as the standard command line setup within the OS. In order to do this you need to edit the interfaces file using **sudo nano /etc/network/interfaces** and then type in the following code, replacing ssid and password with the actual values:

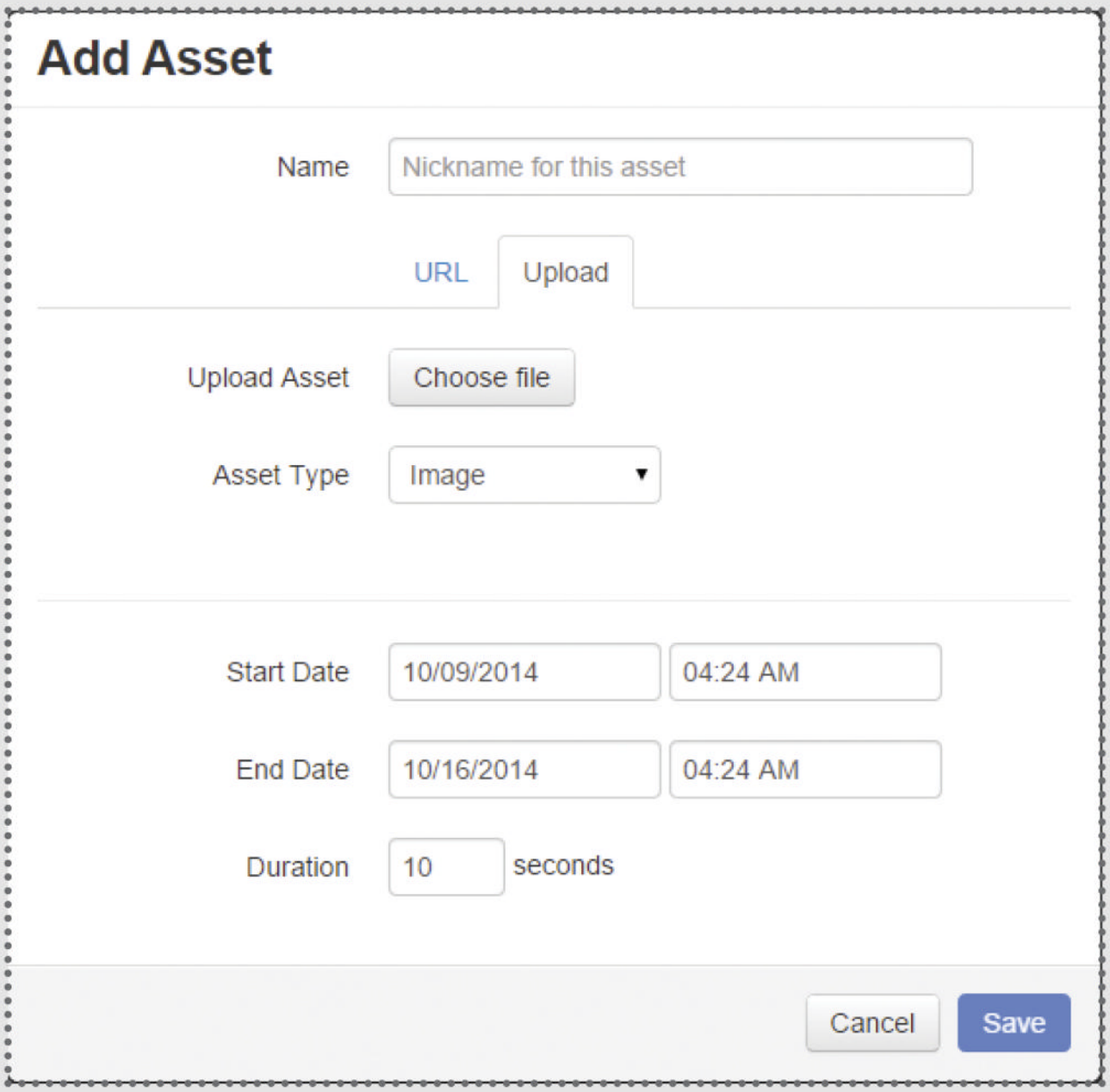
```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
auto wlan0

iface wlan0 inet dhcp
    wpa-ssid “ssid”
    wpa-psk “password”

iface default inet dhcp
```

Then exit and save by hitting Ctrl+X, then Y and then finally the Return key.

13 Test the connection

The easiest way to test the Wi-Fi connection is to shut down the Raspberry Pi using **sudo shutdown -h now** and remove the wired network connection, then reboot the Raspberry Pi by removing and reattaching the microUSB power connector. If the Wi-Fi connection has worked, you should now see the splash screen with IP address again.

“The easiest way to test the Wi-Fi connection is to shut down the Raspberry Pi and remove the wired network connection, then reboot the Raspberry Pi”



Left You can set up your photo frame to display different images at different times of day

14 Upload pictures to Screenly

Once again, you will need to visit the Screenly OSE web interface by entering the IP address into another computer. Since you are now using a wireless connection, the IP address may be different to the previous one. You need to select the 'Add Asset' option at the top right-hand side, which should launch a pop-up options screen. Select Image from the drop-down box and you then have the option of either uploading the image or grabbing it from a URL using the corresponding tabs. Enter the start date and end date of when this image should appear, and how long it should appear on screen for, then press Save. Repeat this step for each of the pictures.

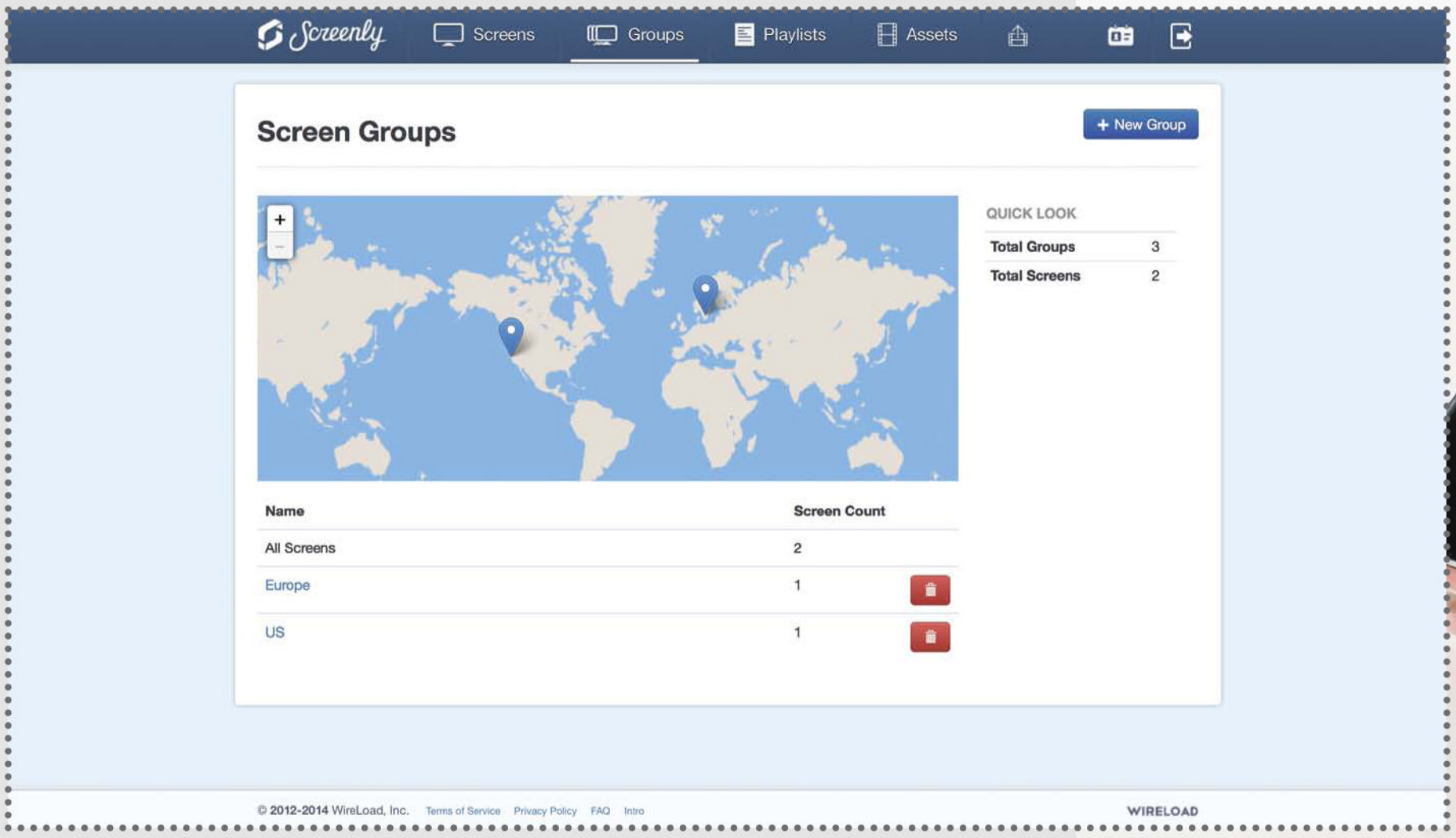
15 Test with video and more

Pictures are great, but Screenly also allows you to display videos (with audio if you wish) and web pages, which really is a huge benefit. This is perfect if you want to enhance your digital photo frame even further or perhaps display the local weather and news to keep yourself informed. Plug in your speaker – we would recommend The Pi Hut portable mini speaker (available from **http://bit.ly/1xEpBNZ**).

16 Place in situ and enjoy!

Once you have got Screenly all set up and loaded all of your favourite pictures and videos onto it via the web interface, it is now time to enjoy the fruits of your labour! Mould the spider stand (if you have one) into shape by taking the middle two legs at the top and bending them downwards and backwards. Then

Below Screenly Pro can manage multiple screens and has cloud storage too



spread the front-middle legs a bit wider to give a good base and shape the outer legs at the top and bottom to support the screen. You are then ready to give it its permanent home – our expert's is on the mantelpiece over the fireplace!

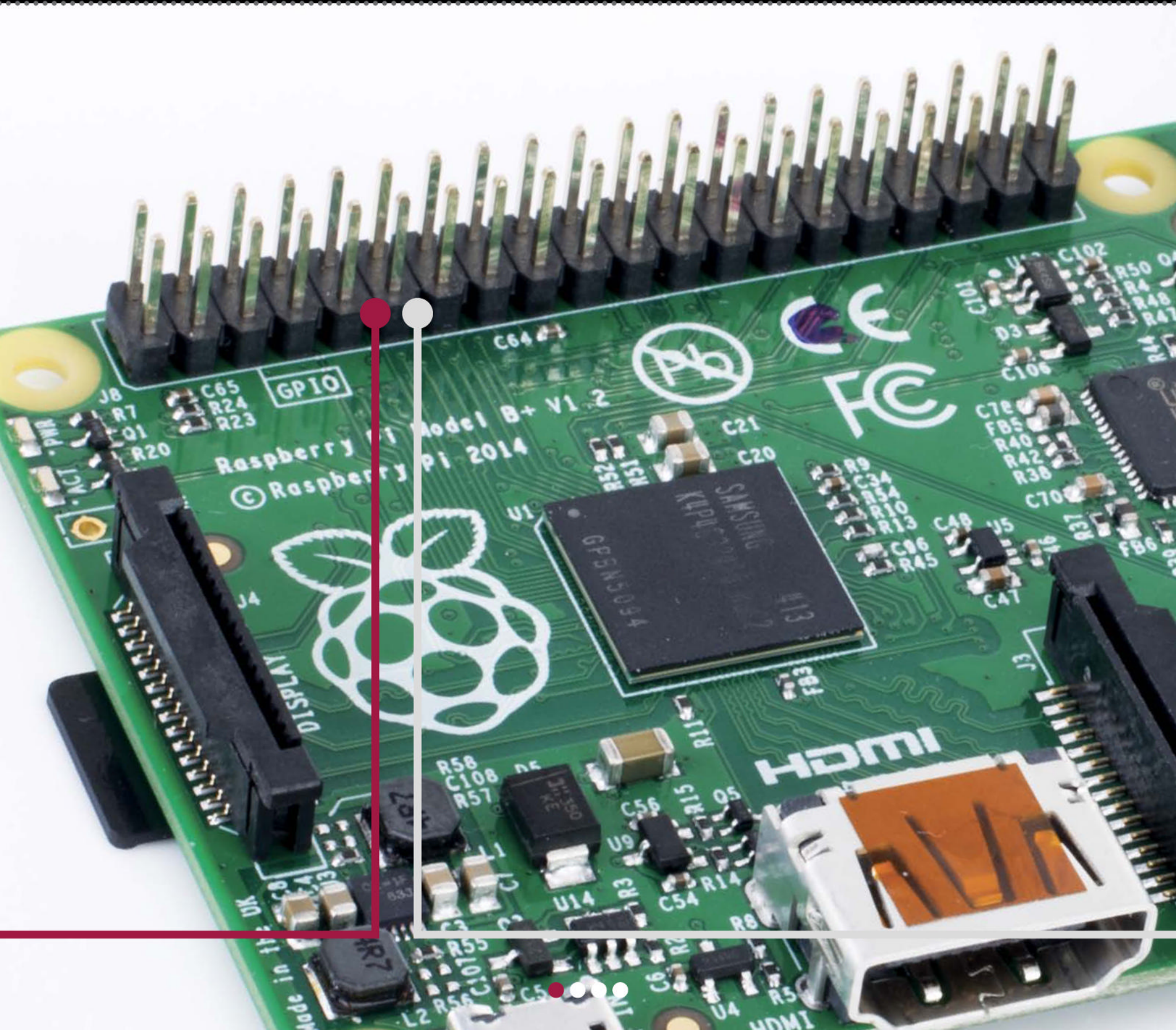
17 Other project ideas

In this tutorial we have looked at just one fairly basic application of Screenly and the HDMIPi. You could use this powerful open source software to run your digital signage empire, share screens in schools and clubs, or as a personal dashboard using a suitable web page. Whatever you make, please don't forget to take pictures and send them to us!

Below Once fully configured, load your pictures and video to complete your digital photo frame!

What are the extra GPIO pins?

The B+, A+ and 2B Models have an extra 14 GPIO pins compared to the original A and B – what do they do?



Q So there are 14 more GPIO pins on the Model B+ of Raspberry Pi, and the later A+ and Pi 2?
A Yes, one of the many changes made to the Raspberry Pi when it was upgraded was to expand the GPIO port so that it was capable of doing more.

Q Why would the Foundation do this?
A The GPIO port has become quite popular among hobbyists and makers who want to use their Pi for physical projects. They can essentially become the main computer/controller for a lot of devices and expand what they can do by interacting with the outside world via wireless internet and such.

Q How many GPIO pins does that mean the new Raspberry Pi models have now?
A All in all, there are now 40 pins in the GPIO port, all of them usable by the Raspberry Pi.

Q Is this the same amount as the Compute Module that was released before the B+?
A Ah no, the Compute Module's IO board has a lot more pins than the Model B+. It has two banks of 100 pins each, meaning it has a total of 200 pins to play around with. It would probably look like a curbside telephone routing cupboard with all the pins wired up.

Q What new things can you do with the added ports?
A Well, there's technically nothing specifically new other than being able to have more inputs and outputs. All the pins are duplicates of the functions you can already get with the original Model B and Model A.

"There's technically nothing specifically new other than being able to have more inputs and outputs. All the pins are duplicates"

Q Nothing specifically new? What's the point of them all then?

A As we said, you now have more inputs and outputs, which means that you can generally do more with the Pi than you were able to before. There are also a couple more power and ground pins, although they're not quite as important as the normal functions.

Q If there are duplicates, does that mean the pins have been rearranged?

A No – the original 26 are still the same here. The types of pins aren't really grouped together either, which allows you to separate them more easily for all the different parts of your project.

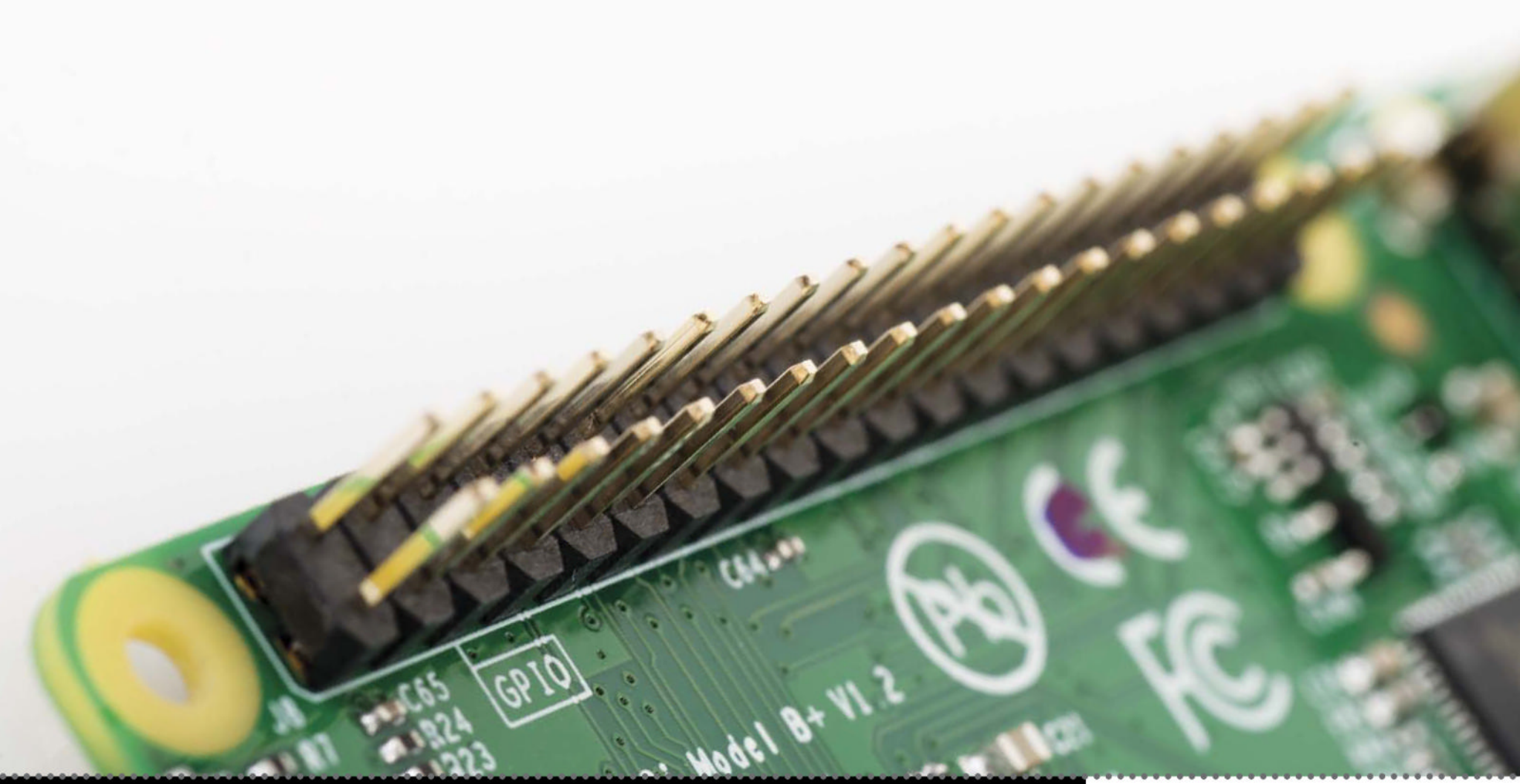
Q What kind of pin types were added?

A Well they've added a MISO (input), MOSI (output) and SCLK (clock) pin, which are used in conjunction with each other as a serial peripheral interface otherwise known as a SPI. As the name suggests it's for peripheral devices, and there's one of each of these pins already on the original Model B. There's also a couple of I2C EEPROM pins that allow for connection to a specific Raspberry Pi add-on that allows for some level of non-volatile memory, the kind used in chips.

Q Wait, that's just five pins – what about the others?

A The rest don't have any specific functions per se, they can all be used as input and outputs roughly the same way. Three of them are ground pins though, and you can't use the I2C EEPROM pins, so there are nine usable input and output pins.

"They've added a MISO (input), MOSI (output) and SCLK (clock) pin, which are used in conjunction with each other as a serial peripheral interface"



Above An additional 14 pins have been included in the GPIO port, but the original 26 are the same

Q Okay then. What other functions do the original GPIO ports include?

A The others include RXD and TXD pins for serial data communication, an extra clock pin, I2C pins for various data input, output and slave select lines for choosing specific chips.

Q Can all of these be used as GPIO pins or are some off limits?

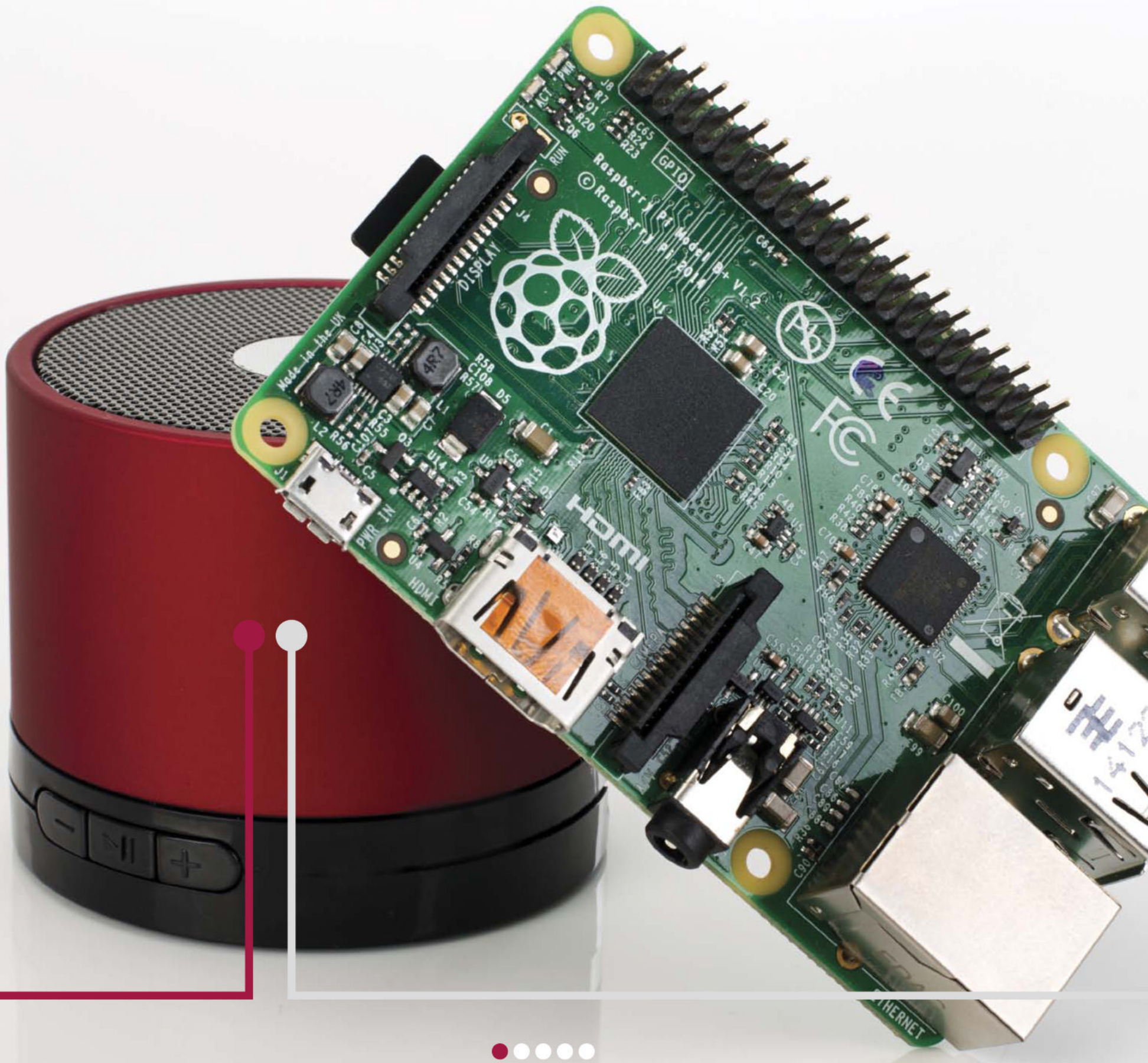
A Aside from the power and ground pins, all of these pins are usable in a normal way instead of the extra functions. You can't really use them both ways at the same time unless you have some physical switching system in place.

Q How can I control the extra pins?

A The same way as before – the necessary packages for Python, the command line and other programming language modules have been updated to allow access to the new pins no problem.

Set up a wireless AirPlay speaker

Combine a Pi, a Wi-Fi dongle and your speakers with an amplifier to create a wireless stereo system



Every home needs a good speaker system, and they're even better if they are accessible to multiple devices over the wireless network. AirPlay uses Apple tech that was reverse-engineered in 2011, which means that third-party devices can now participate in the fun. AirPlay allows any Apple device to broadcast whatever is coming out of its speakers to an AirPlay receiver (in this case, our Pi). There is a way to send audio from PulseAudio to AirPlay receivers, as well as an application for Android called AirAudio. Sadly, the Android application requires root to access the audio of other applications running on the device. To keep things simple, we're going to use an iPad as the test client.

THE PROJECT ESSENTIALS

Latest Raspbian image
raspberrypi.org/downloads

Wireless (or wired) network

USB wireless dongle

Speaker system

01 Set up Wi-Fi

Connect everything to your Raspberry Pi and power it up. Log in to the Raspbian system with the username pi and the password raspberry. Then, make sure your wireless interface is working by typing **iwconfig**. You should see an entry there for wlan0. If you look in /etc/network/interfaces, there should already be a section that looks like this:

```
allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

If it doesn't, then add it at the end. This configuration makes the wireless adapter connect once the network in wpa_supplicant.conf is in range. This means the only thing left to do is to edit the wpa_supplicant.conf file, which needs your SSID (wireless network name) and the passphrase. You can do this with:

> "There is a way to send audio from PulseAudio to AirPlay receivers, as well as an application for Android called AirAudio. Sadly, the Android application requires root"

```
sudo bash -c "wpa_passphrase my_network
my_passphrase >> /etc/wpa_supplicant/wpa_
supplicant.conf"
```

…which will result in a wpa_supplicant.conf file that looks like this:

```
ctrl_interface=DIR=/var/run/wpa_supplicant
GROUP=netdev
update_config=1
network={
    ssid="my_network"
    #psk="my_passphrase"
    psk=17661426f1af334010ad2058d8b8f583ec501....
}
```

The easiest way to get things going is to reboot the Pi with **sudo reboot**. Once your Pi is back up, check that you have an IP address on wlan0 with **ip addr**.

02 Install Shairport dependencies

These instructions are based on a write-up by Drew Lustro (**http://drewlustro.com**). We need to compile Shairport, but before we can do that we have to install its dependencies first:

```
sudo apt-get update; sudo apt-get install git
libao-dev libssl-dev libcrypt-openssl-rsa-perl
libio-socket-inet6-perl libwww-perl avahi-
utils libmodule-build-perl libasound2-dev
libpulse-dev
```

Now we have to compile a Perl module called Net::SDP.

```
cd perl-net-sdp/
git clone https://github.com/njh/perl-net-sdp.
git perl-net-sdp
sudo bash -c "perl Build.PL; ./Build; ./Build
test; ./Build install"
cd ..
```

"We need to compile Shairport, but before we can do that we have to install its dependencies first"

03 Compile and install Shairport

Now we get to compile and install Shairport, which might take a minute or so:

```
git clone https://github.com/abrasive/shairport.git
cd shairport/
./configure; make; sudo make install
```

After this, the Shairport binary will be installed but it won't be started at boot, which is unhelpful. To fix this, we need to install the init script:

```
sudo cp scripts/debian/init.d/shairport /etc/init.d/  shairport
sudo chmod 755 /etc/init.d/shairport
sudo update-rc.d shairport defaults
```

Finally, Shairport needs its own user that is a member of the audio group. You can create one with:

```
sudo useradd -g audio shairport
```

04 Change the hostname

Shairport will show up as whatever the hostname of the system is. If you don't want the name to be raspberrypi, then you can change it like so:

```
sudo bash -c "echo PiPlay > /etc/hostname"
sudo hostname PiPlay
sudo sed -i 's/raspberrypi/PiPlay/g' /etc/hosts
```

05 Set the default sound card

If you have a USB sound card then you'll want to make that the default. You'll also want to make sure the volume is set to 100 per cent so that your amplifier is getting a nice signal. Use **aplay -L** to list the audio devices on your system. Our expert's sound card had an entry like this:

```
front:CARD=U0x41e0x30d3,DEV=0
      USB Device 0x41e:0x30d3, USB Audio
      Front speakers
```

To set that card as default, you do:

```
sudo bash -c ‘echo pcm.!default
front:U0x41e0x30d3 > /etc/asound.conf’
```

You can set the volume by going into alsamixer, pressing F6 to select the appropriate sound card and then using the up and down arrows to change the volume. Once you are done, you can make the changes the default with **sudo alsactl store 1**, where 1 is the card number. If you are using the built-in sound card then you should use 0 instead.

Finally, you can test the sound card is working with **speaker-test -c 2**, which sends a test signal to the left speaker and then the right speaker. You can exit the test with Ctrl+C.

06 A final reboot

After this reboot, Shairport should start automatically. To test it out, get your client and see if it can find the PiPlay. On an iOS device, slide up from the bottom to bring up Control Center, tap the AirPlay button and select PiPlay. It might take a couple of seconds to buffer, but once it’s playing it should be fine.

07 Make it one unit

This is the freestyle part of the tutorial. Our expert decided to turn his setup into a single unit, with the T-class amp and the Pi stacked on top of his speaker. Obviously, it could always be tidier, but that would involve cutting cables and we didn’t have any spares.

Remotely control your Pi

Use a web interface to control your Pi and employ it as a file server or media centre from any online device

Not everyone uses the Raspberry Pi while it's hooked up to a monitor like a normal PC. Due to its size and portability, it can be located almost anywhere that it can be powered and it's widely used as a file server, media centre and for other nontraditional applications. Some of these uses won't easily allow access to a monitor for easy updates and maintenance. While you can always SSH in, it's a bit slower than a full web interface that allows for custom commands and a view of the Pi's performance. We're using software called RaspCTL, which is still in development, but works just fine for now.

THE PROJECT ESSENTIALS

Raspbian set to command line

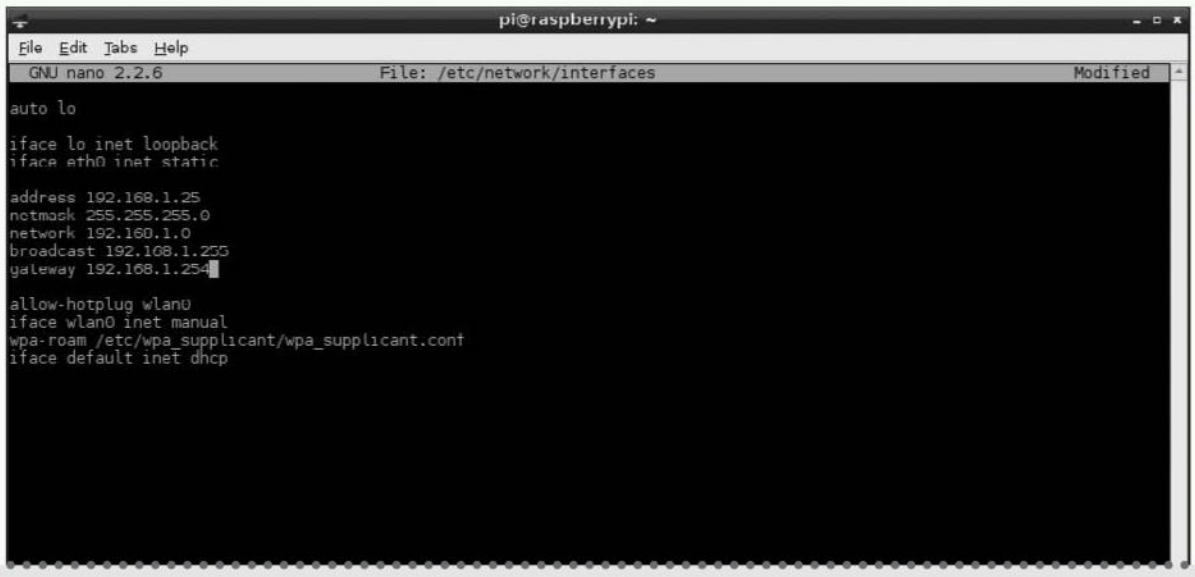
RaspCTL

Internet connection

"A full web interface allows for custom commands and a view of the Pi's performance"

01 Update your Pi

To make sure the Raspberry Pi works as best it can, you'll need to update Raspbian. Do this with a **sudo apt-get update && apt-get upgrade**, followed by a firmware update with **sudo rpi-update**. Finally, if you're booting to LXDE, enter **raspi-config** and change it to boot to command line to save power.



02 Edit the IP

For everything to work more easily, you should set the Raspberry Pi to have a static IP of your choice. To do this, edit the networking config by using:

```
$ sudo nano /etc/network/interfaces
```

...and change iface eth0 inet dhcp to **iface eth0 inet static**.

03 Set up a static IP

Add the following lines under the iface line with your relevant details:

```
address 192.168.1.[IP]
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.[Router IP]
```

04 Ready to install

You'll need to grab the public keys for the software we're going to install by using the following commands. The first will take just a moment to download the software, while the other quickly installs it:

"For everything to work more easily, you should set the Raspberry Pi to have a static IP of your choice. To do this, edit the networking config"

```
$ wget debrepo.krenel.org/raspctl.asc
$ cat raspctl.asc | sudo apt-key add -
```

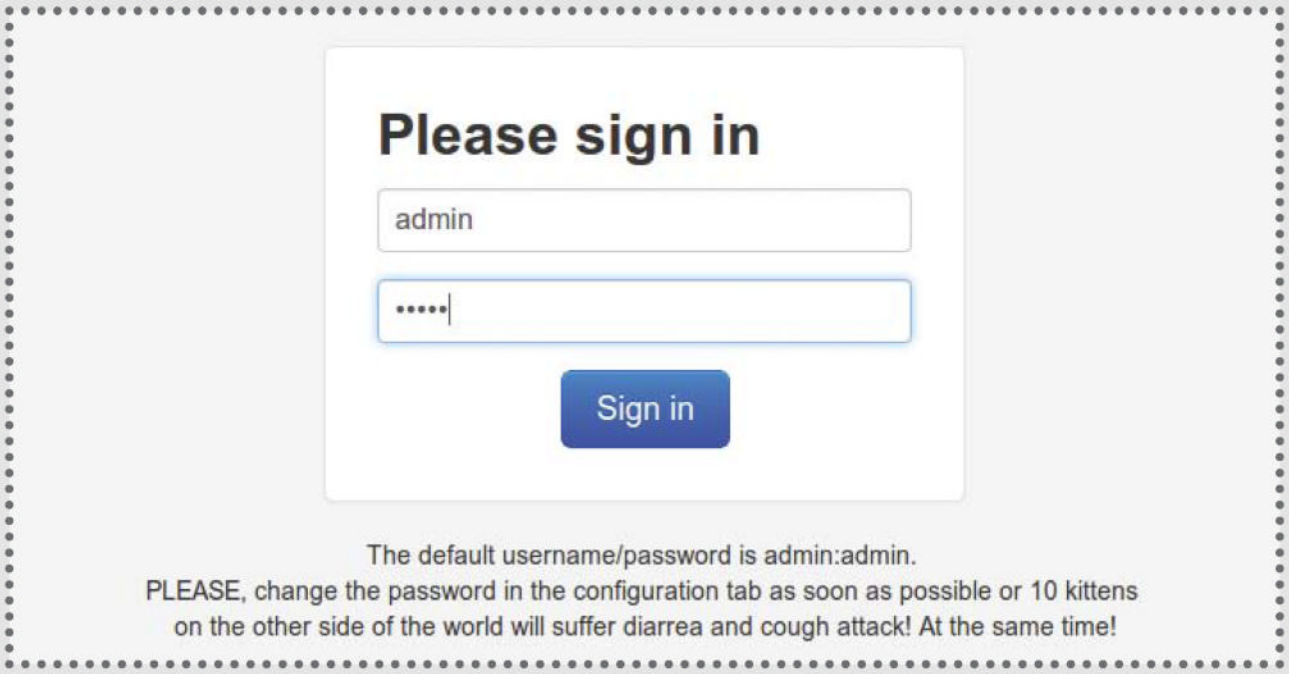
05 Add the repository and install

Add the repository to the source's file with the following command:

```
$ echo “deb http://debrepo.krenel.org/ raspctl
main” | sudo tee /etc/apt/sources.list.d/
raspctl.list
```

…and finally install the software with:

```
$ sudo apt-get update
$ sudo apt-get install raspctl
```



06 Access your Raspberry Pi

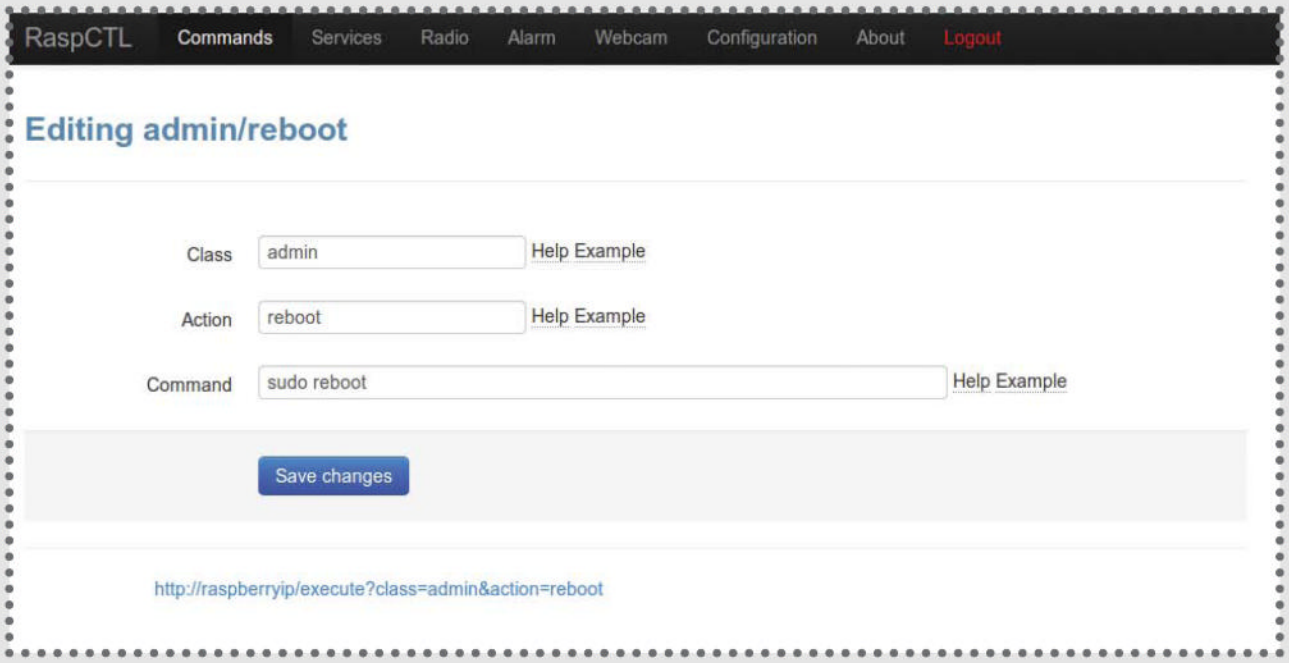
Now the software is installed you can start to access your Raspberry Pi from anywhere on your network. To do this type the following into your address bar, with the IP being the one we set up earlier:

```
http://[IP]:8086
```

“Now the software is installed you can start to access your Raspberry Pi from anywhere on your network”

07 Change your password

The default username and password is admin for both fields, and you should make sure to change that before doing anything else. Go to Configuration along the top bar and find the Authentication field at the bottom of the page. Input the original password (admin), followed by your new passwords. The username will remain as admin.



08 First command

Go to Commands on the top bar to begin creating commands to run. Here you'll need to add a class – a user-defined way to filter your commands that won't affect the way it's run – a name for the command and the actual command itself. The commands won't necessarily run from the pi user unless you tweak the config files.

09 More functions

The web interface has a few extra functions apart from running commands, such as the ability to view the webcam and connect to radio services. Play around to see what suits you. Updating the software every so often will also help you make sure it keeps working.

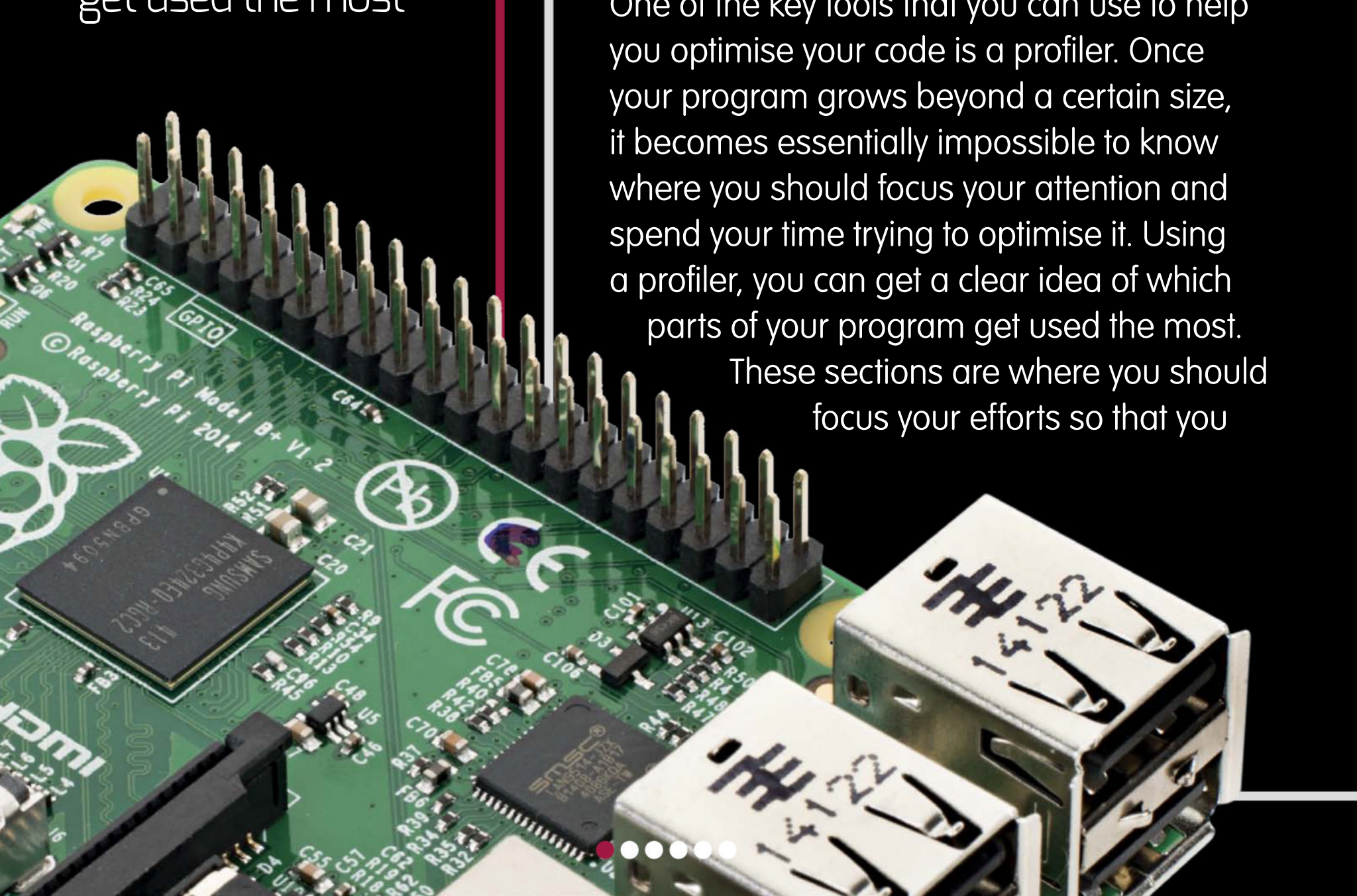
"Go to Commands on the top bar to begin creating commands to run. You'll need to add a class, a name for the command and the actual command itself"

Optimising code for your Raspberry Pi

In any low-resource system, you need to make maximum use of what is available. Profiling helps

"Using a profiler, you can get a clear idea of which parts of your program get used the most"

One of the issues when trying to develop programs to run on a Raspberry Pi is to be as efficient as possible. This is true for any system, but even more so for systems with limited resources. One of the key tools that you can use to help you optimise your code is a profiler. Once your program grows beyond a certain size, it becomes essentially impossible to know where you should focus your attention and spend your time trying to optimise it. Using a profiler, you can get a clear idea of which parts of your program get used the most. These sections are where you should focus your efforts so that you



get the biggest impact. You should be aware that profiling is different from benchmarking. Your program will take longer to run under a profile because of the additional overhead tasks. The important information is not how much total time is being used in different sections of code, but what percentage of the time is being spent in each section. With compiled languages, like C or C++, profilers are external programs that need to link in to your running code and poke around to see what is happening. With Python, profilers are modules that you load just before you run your code. This is one of the great advantages of coding in an interpreted language. You can squeeze other code in to handle monitoring activities. This month, we will take a look at two of the most popular profiling modules and see how they can help optimise your code.

The first profiling module we will look at is cProfile. This module is actually a C extension. Being written in C, it is optimised to minimise the amount of overhead involved, making it very good for any programs that have a longer run time. This is actually the recommended option for most people. You load the cProfile module by using the following command:

```
import cProfile
```

You can now run Python code through the profiler. The call requires the same format you would use if you were to 'exec' some code. The easiest thing to do here is to have your code packaged as a function. As a quick and dirty example, let's write a function that calculates the cube of all of the integers up to some limit:

```
def looper(size):
```

```
    for a in range(size):
        a*a*a
```

You can then run this function inside the profiler using this command:

```
cProfile.run('looper(1000000)')
```

The output you get looks like that in Fig. 01, below.

You get six columns of output from a profile run. The first column is the number of times that particular function is called. The second column is the total amount of time used by that function, minus any time spent in calls to subfunctions. You should notice that the time values are given to a precision of 0.001 seconds. This is because the profiler only records time to within the nearest clock tick. On most machines today, the clock tick is set to 10ms. In most cases, this should be good enough. If you are worrying about functions that take less than 10ms to run, you probably need to be working closer to the metal then you can get with Python. The third column is the amount of time used per call. The fourth and fifth columns give the same times, except they include the amount of time

"You should notice that the time values are given to a precision of 0.001 seconds. This is because the profiler only records time to within the nearest clock tick"

Fig. 01

```
    4 function calls in 0.878 seconds
  Ordered by: standard name
```

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.668	0.668	0.878	0.878	<ipython-input-.....
1	0.000	0.000	0.878	0.878	<string>:1(<module>)
1	0.000	0.000	0.000	0.000	{method 'disable' of....
1	0.210	0.210	0.210	0.210	{range}

spent in calls to subfunctions. The last column is the location of the relevant function call. Some of these can get quite long, so in our example the longer lines have been truncated a bit for space. You may also see two numbers for a given line in the first column. These cases are when a particular function is recursive. In the code listing, you can see a classic factorial function and the output from a profiling run.

While this is great when playing around in an interactive session, what can you do if you want to keep the profiling data for later analysis? The run function of cProfile will accept a second parameter containing a filename where all of the profiling data will get saved off to. So you could call:

```
cProfile.run('myfunction()', 'prof_data')
```

The profiler will then save all of the collected data into the file 'prof_data'. Now you can have a permanent record of your progress in trying to optimise your code. You can use the pstats module to load the data and play around with it. You could load the data, strip any extraneous path information from the module names and sort it with:

```
import pstats
p = pstats.Stats('prof_data')
  p.strip_dirs().sort_stats(-1).print_stats()
```

These techniques work fine when you are still in the development stage, but what can you do if you inherit some modules? How do you decide where to start? You can run full scripts through cProfile from the command line by using:

```
python -m cProfile -o prof_data myscript.py
```

This will run the profiler on the file 'myscript.py' and save the profiling data to the file 'prof_data'.

The second popular profiling module is profile. This module provides the same interface as cProfile, but is written purely in Python. This means that there is a good deal more overhead involved, so the run time for a profiling run may be significant. But, since it is pure Python, it is open to being modified easily. This is a great help if you wanted to build more functionality on top of that provide by profile. Also, profile will work on systems where cProfile is not implemented. The one that you decide to use will depend on your specific needs.

With the information that we have covered this issue, you should have the tools you need to be able to figure out where to apply your optimisation skills. So now, you should have no excuses for having slow and bloated code. Go forth and optimise!

"Since profile is pure Python, it is open to being modified easily. This is a great help if you wanted to build more functionality on top of that provide by profile"

The Code PROFILING CODE

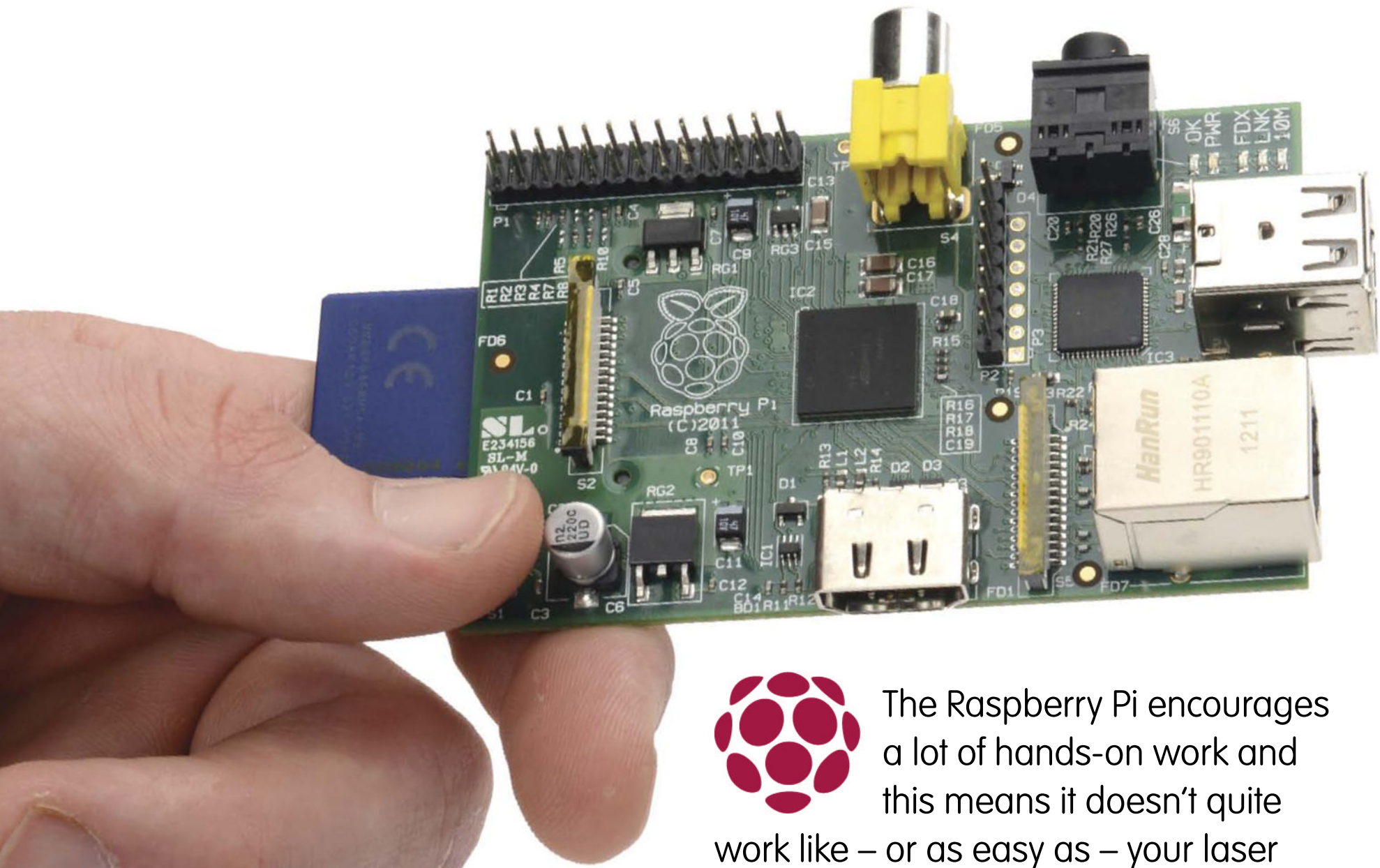
```
# Here we will run through some
# examples for cProfile

# We will define a looping function
# that calculates the cube of a range
# of numbers to chew up CPU cycles
def looper(size):
    for a in range(size):
        a*a*a

# Start by importing cProfile
import cProfile

# Run looper for a large size
# and profile it's behaviour
cProfile.run('looper(10000)')

# We'll setup a "bad" factorial
# function to look at recursion
def myfact(a):
    if a == 0:
        return 1
    else:
    return a * myfact(a-1)

# We'll run this and save it off
# to a file for later
cProfile.run('myfact(50)', 'prof_data')

# Now we will load it in to pstats
import pstats
p = pstats.Stats('prof_data')
p.strip_dirs().print_stats()
```

"Profile will work on systems where cProfile is not implemented. The one that you decide to use will depend on your specific needs"

Talking Pi

Join the conversation at...

@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk



The Raspberry Pi encourages a lot of hands-on work and this means it doesn't quite work like – or as easy as – your laser focus-tested smartphone interface, or even a normal computer. So we're answering your burning Raspberry Pi questions each issue – get in touch with us on Twitter, Facebook or by email.

Can I use my Raspberry Pi for cosplay?
Sam via email

It's actually very well suited for cosplay use – the low power requirements, size and GPIO pins means you can fit it into small spaces and have it sufficiently powered to control LEDs for cool light effects or actuators and servos for moving parts. You'd need to power those separately though, but the automation and programming of the parts can be made very easy with the Raspberry Pi, a bit of Arduino and a decent amount of researching and testing. Think the original *Iron Man* movie.

Is the Raspberry Pi waterproof? I want to use it outside.
Claire via Facebook

Absolutely not! It's a bare circuit board and it's very susceptible to shorts and corrosion from the elements. There are solutions you can use to waterproof it, such as special PICE waterproof cases, and you could build your own rain-proof enclosure fairly easily (as long as you don't plan to submerge it, that is). The Pi camera can also fit in the PICE if you're looking to do video as well.

Keep up with the latest Raspberry Pi news by following @LinuxUserMag on Twitter. Search for the hashtag #RasPiMag

JUST A SCORE
WHAT'S YOUR JUST A SCORE?

Have you heard of Just A Score? It's a new, completely free app that gives you all the latest review scores. You can score anything in the world, like and share scores, follow scorers for your favourite topics and much more. And it's really good fun!

How durable is the Raspberry Pi?

Hywel via Twitter

The Raspberry Pis are made very well – although the actual cost of them is quite cheap, the parts have not been skimped on. As the Pis are sold by a charity, they don't technically need to make a profit. The board itself, with a decent amount of care, will be fine for many years. As for being in more physical environments with a lot of movement, it's physically quite robust and won't break very easily, but we'd recommend testing before using it in such strenuous environments.

Can I automatically back up the contents of my Raspberry Pi periodically or continuously?

James via Facebook

Yes, you can. There are a number of ways you can attempt to do it – firstly you could try and set up Dropbox on the Pi; it's available for it with a bit of tweaking and keeping specific files in a folder/directory that's continuously syncing. That's the easy way, and it can eat up data if that's a problem. Otherwise, you can set up a 'crobjob' that will automatically, periodically, move and copy files around, such as to another computer on the network.

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

PROGRAM ROBOTS

Get this issue's source code at:

www.linuxuser.co.uk/raspicode